ST443 Group Project

Candidates: 36320 (25%), 34500 (25%), 33552 (25%), 35190 (25%)

December 2021

Abstract

Real world data analysis of features, genre and popularity of pieces of music. Estimation of conditional dependency structure of sparse and non sparse graphical models using Lasso based approaches, in particular, Node-wise Lasso 1, Node-wise Lasso 2 and Graphical Lasso.

Contents

Chapter 1

Real World Data

1.1 Introduction

A piece of music is often considered as a work of art, a medium with which artists can express themselves and let their creative spirits roam wild. It therefore seems hard to imagine the existence of a link between abstract expressions of art and data analysis, a relationship which this paper will seek to explore. Spotify as a top music platform occupying over 36% market and having over 100 million subscribers, has one of the biggest music information databases. Therefore through the use of a Spotify songs database, we will dig deeper into the structure of songs, aiming to find answers to the following questions:

1. Which music elements can be used to predict the genre of songs?
2. Which elements can be used to determine whether a track will be popular? Could our analysis be of benefit to music producers?
3. What is the relative importance and direction of influence on popularity of each element?

1.2 Data Description

The original data set is the $Spotify\_songs$ data set, a data set of over 30,000 songs with 12 audio features and 11 variables related to track information (details of each column can be found in Appendix A.1 A.2). In order to tackle our questions, we derive two data sets from the original: a Popularity data set and a Genre data set.

1. **Popularity Data set:** The Popularity data set is used to predict the degree of popularity of different songs. It has 12 audio variables and 3 variables relating to track information. The genre variable is one hot encoded and also added to the data set. Thus, its final size is 32833 *20. Details of each variable can be found in Figure A.3.
2. **Genre Data set:** To create the Genre Data set, we extract the column *playlist_genre* containing 6 different genres. Then we extract columns *danceability*, *energy*, *loudness*, *speechiness*, *instrumentalness*, *liveness*, *valence* and *tempo*, all of which are variables

closely connected to our response variable *playlist_genre*. Details of each variable can be found in Figure A.4.

1.3 Methodology

1.3.1 Binary Classification (Appendix A.3)

Following a binary classification approach, we classified the numerical value of *track_popularity* into 2 classes: 'popular' and 'unpopular'. This variable is stored as a new column in the data set as the variable *Popular*. The Popular column takes the value of 'popular' when *track_popularity* exceeds 50, and unpopular otherwise.

We first start with the pre-processing of the Popular data set, in which null elements and duplicates are removed. Then we divide the data set into a training data set and a test data set using a ratio of 7:3. Later, we use logistic regression, LDA, KNN, decision tree, bagging, random forest tree and boosting, as methods to classify tracks into 'popular' or 'unpopular. The accuracy on testing data is used to evaluate our models.

1.3.2 Multi-class Classification (Appendix A.4)

The playlist genre, our response variable, is divided into six categories: EDM, *Latin*, *Pop*, *R&B* and *Rock*. There are eight related variables that can determine the genre of a song. Here, we use some models to achieve a multi-class classification. We split the data set into two parts: the training set and the test set. After training the models using the training data, we made predictions in order to obtain the prediction accuracy rate as well as the AUC from the ROC curves. The machine learning models used are as follows: logistic regression, LDA, QDA, KNN, trees & pruned trees, boosting, bagging, random forest and support vector machine.

1.3.3 Regression Method (Appendix A.5)

To better understand the direction of influence of these predictors on the track popularity, we also do regression with response variable *track_popularity*. Variable selection was performed, to determine the optimal model based on evaluation metrics, specifically RSS, BIC, C_p, Adjusted R^2. Furthermore, ridge regression and lasso regression was used to better understand the importance and the direction of influence of various predictors.

1.4 Results

1.4.1 Binary Classification

As we can observe from Table 1, random forest and bagging models have the highest prediction accuracy at around 0.7. They have the highest True Positive Rate (TPR) and a relatively low False Positive Rate (FPR). The accuracy of each model is listed in Table A.1 of the Appendix. By looking at the importance figure developed using the bagging and random forest method (Figure A.6), we get that the feature *loudness*, followed by *acousticness*, *energy* and *duration*, are the most significant audio features when predicting the degree of popularity

of a song. On the other hand, the boosting model indicated that *tempo* is in fact the most important feature, followed by *duration*, *loudness*, and *acousticness* Figure A.7). Given that the *loudness*, *acousticness* and *duration* variables appear as one of the most important features in both methods, one could conclude that these are the elements that have the greatest influence on popularity.

1.4.2 Multi-class Classification

We can easily see that the model of bagging and random forest far outperform other methods. This means that bagging and random forest models can provide more accurate results for predicting the genre of a work. The prediction accuracy these two models can achieve are 0.8985442 and 0.9066431 respectively. The accuracy of each model is listed in Table A.2 of the Appendix. Moreover, from the bagging importance plot (Figure A.12) and the boosting importance plot (Figure A.13), one can see that the *tempo* is the most importance predictor, followed by *speechiness*, *danceability*, *energy*, *valence* and *loudness*. Therefore one could consider the tempo of the song to be the most important variable affecting the genre of a track.

1.4.3 Regression Methods

1. **Best subset selection:** Here, we use the forward approach to obtain the best model. From the plot which shows the relationship of various evaluation metrics and the number of variables (Figure A.19), we can derive that most of the variables are quite significant in the regression model. More importantly, by looking at the coefficients we get that audio features like *energy*, *speechiness*, *instrumentalness*, *liveness* and *duration* have a negative influence on songs' popularity. In contrast, higher *danceability*, *loudness* and *acousticness* can make songs more attractive. What's more, we find that the electronic dance music genre has the largest negative effect on a song's popularity, while the rock genre has the lowest negative effect on the popularity. The outputs are listed in Figure A.16.

2. **Ridge Regression:** We used cross-validation to find the best lambda, and in contrast to best subset selection, it shows that rock, pop and latin music have a positive effect on the popularity of songs, while electronic dance music, R&B and rap have a negative effect. The output is listed in Figure A.17.

3. **Lasso Regression:** For the lasso regression, we use cross validation to get the best λ first. The results are similar to those of ridge regression. The output is listed in Figure A.18 .

1.5 Discussion

Binary Classification: In this case, we determined the threshold of a song being defined as popular as *track_popularity* exceeding 50, which is a little above the median of popularity. This definition however is very flexible, and without any reference as to how to define a 'popular' song, it's easy to miss-classify songs. To therefore improve the model and increase the robustness of our investigation, an improvement would be construct a survey and search

some reference online, for example on how other music platforms classify songs as being popular, to get a more scientific threshold for popular.

Multi-class Classification: Moving on to the multi-class classification models, we find that random forest and bagging models have the highest prediction accuracy and thus one could consider them as the optimal models for classifying genres. One possible reason for the similarity of results of these models, is the large dimensions of the data set. The difference between the two models is the parameter value of mtry, which however doesn't greatly influence the final result, when using large volumes of data. Secondly, we can find that the random forest model can handle high-dimensional data well without feature selection (because feature subsets are randomly selected).This can also explain why other classification models do not have high accuracy when there are too many variables.These models however may need to be used with caution. Taking one step back and looking at the concept of genre of a piece of music, it is no secret that there may be some grey area relating to how the genre of a piece is determined. The decision is very subjective and a large amount of songs may fall into an overlap of different genres, possibly influencing the validity of our results.

Regression Methods: In the regression analysis, one potential improvement would be to make a choice of the tuning parameters using the one standard error to avoid over-fitting. Finally, there exists some imbalance between the music genres, therefore one could consider further pre-processing the data to obtain more balance.

1.6 Conclusions

1. **Which music elements can be used to predict the genre of songs?**

   Using the bagging and boosting outputs, one can determine that *tempo* is the most important predictor which can affect the genre of a piece of music, and it's the variable which leads to the largest reduction in the Gini index. We could have made a more accurate analysis of the impact of variables through PCA and feature selection, but considering time and the fact that we already used the random forest method in our analysis, we chose not do so.

2. **Which elements can be used to determine whether a track will be popular? Could our analysis be of benefit to music producers?**

   Through the random forest/bagging approach of predicting whether a song is popular or not, we get that the most significant variables determing popularity are *loundness*, *tempo* and *speechiness*. With this model, we get an accuracy rate of 71% with a true positive rate of approximately 70% and less than 30% false positive rate, allowing us to predict popularity better than all the other models we have considered. As a result, this would allow music producers to obtain a first identification of the popularity of a song in development, thus greatly reducing their work when determining whether to release a track or not.

3. **What is the relative importance and direction of influence on popularity of each variable?**

   As shown in Figure 3.9, we know that *loudness*, *tempo*, *speechiness*, *duration*, *energy*, *acousticness*, *danceability*, *valence*, *liveness*, *instrumentalness* and *years*, play a huge role in the prediction of popularity. Finally, the output of ridge regression and lasso regression indicate that *energy*, *speechiness*, *instrumentalness*, *liveness*, *valence* and *duration* all have a negative influence on the popularity of a song.

Chapter 2

Estimation of graphical models using lasso-related approaches

2.1 Introduction to the Graphical Model

The graphical model refers to an undirected graph representing the conditional dependency structure of variables of interest.

Given a vector of p random variables $X = (X_1, X_2, ..X_p)$, the graphical model is of the form $G = (V, E)$,where $V = \{1, 2, .., p\}$ represents the vertex set and $E = \{e_{ij}, 1 \leq i \neq j \leq p\}$ represents the edge set. Each node in the vertex set corresponds to one of the p components of X, and each edge e_{jl} in the edge set, is an indicator of the presence of dependency/covariance of vertex j and vertex l, conditional on the remaining vertices. More explicitly, vertex j and vertex l are connected by an edge e_{jl} if and only if $c_{jl} \neq 0$ where $c_{jl} = Cov(X_j, X_l | X_k, 1 \leq k \leq p, k \neq j, l)$.

Thus the edge set E of the graph $G = (V, E)$ is in fact of the following form:

$$E = \{(j, l) : c_{jl} \neq 0, 1 \leq j, l \leq p, j \neq l\} \tag{2.1}$$

2.1.1 Gaussian Graphical Model (GGM)

Let us now consider a sub field of the Graphical model, namely the Gaussian Graphical Model (GGM), where an extra assumption is added, as described by Fan et al. (2020) and Angelini et al. (2021). In particular, it is assumed that the multivariate random vector X the model is describing, follows a multivariate gaussian (normal) distribution with $\mu = 0$ and covariance matrix $\Sigma \in \mathbf{R}^{p \times p}$. i.e $X = (X_1, X_2, .., X_p) \sim N(0, \Sigma)$

This extra assumption allows for a straight-forward estimation of the edge set (2.1) leveraging Lasso methods, specifically node-wise lasso and graphical lasso, indicating the value of GGMs.

2.2 Node-wise lasso Approach

The node wise lasso approach of estimating a sparse graphical model was first introduced by Meinshausen and Bühlmann (2006). To understand the approach one must first consider the concept of a neighbourhood of a node. Given variables $X_1, X_2, \dots X_p$ and corresponding nodes $V = \{1, 2, ...p\}$, the "neighbourhood ne_j of a node $j \in V$ is the smallest subset of $V \setminus \{j\}$ such that X_j is conditionally independent of all remaining variables". Therefore in essence, the neighborhood of node j are all the nodes with which node j is connected to by an edge (nodes with which $e_{jl} \neq 0$ for $l \in V, l \neq j$). It is shown in the aforementioned paper that fitting a lasso, regressing X_j for node $j \in V$ on the remaining variables X_l $(l \in V, l \neq j)$, allows for a neighborhood estimate (parametrized by λ) of node j to be identified. It is defined in the following way: $ne_{j^\lambda} = \{l \in V : \hat{\beta}_{jl^\lambda} \neq 0\}$ where $\hat{\beta}_{jl^\lambda}$ is the lasso estimator for β_{jl} with λ as tuning parameter.

As a result, if $\hat{\beta}_{jl^\lambda} \neq 0$, node $l \in ne_{j^\lambda}$ and so nodes j and l are estimated to be connected. Fitting the aforementioned lasso regression for each of the nodes with a certain choice of tuning parameter λ for each regression, one can estimate the neighborhood of each node and thus in turn the edge set 2.1 using either the 'joint' or 'or' rules, which correspond to node-wise lasso 1 and node-wise lasso 2 approaches respectively (Friedman et al., 2010).

Under the node-wise lasso 1 approach, two nodes j and l for $j, l \in V$, are estimated to be connected if the estimated lasso coefficient of variable X_l with X_j as response and vice versa, are both non zero. The estimate of the edge set (2.1) can thus be defined as:

$$\hat{E}_1 = \{(j, l) : \text{ both } \hat{\beta}_{jl} \text{ and } \hat{\beta}_{lj} \text{ are non zero }, 1 \leq j, l \leq p, j \neq l\}$$

Under the node-wise lasso 2 approach, two nodes j and l for $j, l \in V$, are estimated to be connected if either the estimated lasso coefficient of variable X_l with X_j as response or vice versa, are non-zero. The estimate of the edge set (2.1) can thus be defined as:

$$\hat{E}_2 = \{(j, l) : \text{ either } \hat{\beta}_{jl} \text{ or } \hat{\beta}_{lj} \text{ is non zero }, 1 \leq j, l \leq p, j \neq l\}$$

2.3 Graphical Lasso Approach

Let us now consider an alternative way of estimating the conditional dependency structure of variables; the Graphical Lasso Approach (Mazumder and Hastie, 2012),(Friedman, Hastie, and Tibshirani, Friedman et al.). Under the assumptions of the Graphical Gaussian model in which variables $X = (X_1, X_2, .., X_p) \sim N(0, \Sigma)$, it can be shown that variables X_j and X_l are conditionally independent given the other variables, if and only if the jl_{th} component of $\Sigma^{-1} = \Theta = 0$, where Θ is a sparse matrix called the precision matrix (Yuan and Lin, 2007). As a result, the edge set (1) can be re written in the form

$$E = \{(j, l) : \Theta_{jl} \neq 0, 1 \leq j, l \leq p, j \neq l\}$$

To now estimate Θ, and thereby estimate the network structure through obtaining the estimated edge set

$$\hat{E}_3 = \{(j, l) : \hat{\Theta}_{jl} \neq 0, 1 \leq j, l \leq p, j \neq l\}$$

one can use the graphical lasso.

Under this approach, an l_1 penalty on the off diagonal entries of Θ is added to the negative gaussian log-likelihood of GGMs = log (det Θ) $- trace(\Sigma\Theta)$ where $\bar{X} = \frac{1}{n}\sum_i^n X_i$ and $S = \frac{1}{n-1}\sum_i^n (X_i - \bar{X})(X_i - \bar{X})^T$. The following expression is minimized over all positive definite matrices $\Theta \in \mathbf{R}^{p\times p}$ to provide a sparse estimate $\hat{\Theta}$.

The estimator therefore takes the following form:

$$\hat{\Theta} = \underset{\Theta \succ 0}{\text{minimize}} \text{ -log (det } \Theta) + \text{ trace}(\Sigma\Theta) + \lambda \sum_{j\neq l} |\Theta_{jl}|$$

$\sum_{j\neq l} |\Theta_{jl}|$ denotes the sum of the off diagonal elements of the matrix Θ and λ is a tuning parameter controlling the size of the l_1 penalty/shrinkage. In essence our estimator with $\lambda > 0$ has the effect of regularizing the estimate $\hat{\Theta}$ to ensure it is always invertible, and also controls the sparsity level of $\hat{\Theta}$. In the case where $\lambda = 0$, no shrinkage occurs and the maximum likelihood estimate is obtained. As the value of the λ increases, the sparsity of our estimate increases as well. The choice therefore of the tuning parameter λ plays a significant role in the performance of the estimator defined above and thus must be selected with care (Banerjee et al., 2008).

2.4 Choice of tuning parameter

2.4.1 Node-wise lasso - Finding λ_{opt}

As previously mentioned, the node wise lasso approach uses a collection of lasso regressions to estimate connectivity between different variables of interest. Lasso regression performs L1 regularization, which penalises the magnitude of coefficients, and may have the effect of shrinking some coefficient estimates to zero, dependent on the strength of the L1 penalty. It takes the following form:

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j X_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| = RSS + \lambda \sum_{j=1}^{p} |\beta_j|$$

As stated by Banerjee et al. (2008) "by choosing the regularization parameter in a particular way, the probability that two distinct connectivity components are falsely joined in the estimated graph can be controlled", thus it plays a significant role in the accuracy/consistency of the estimated edge set. The choice of lambda cannot be made lightly, specific techniques are required to determine the optimal tuning parameter, some of which are discussed in this report. We will focus on three techniques:

1. k-fold Cross Validation - One standard error rule and Best λ approaches
2. AIC - One standard error rule and Best λ approaches
3. BIC - One standard error rule and Best λ approaches

As part of this investigation we consider two approaches to choosing the tuning parameter given a specific technique; the best lambda approach and the one standard error rule

approach. Given the best lambda approach, the optimal λ is the λ that minimises the cross validation error, the AIC and the BIC, for techniques 1 to 3 respectively, as lower values of these metrics are more favourable. This however can lead to over-fitting, and thus a model which would not perform well when presented with new unseen data it has not been trained on.

Given the One standard error rule approach, the optimal λ is the λ who's CV error/ AIC value/ BIC value (depending on the technique) lies within 1 standard error of the minimum error or value. This allows for over-fitting to be avoided as a simpler model is chosen, whilst still retaining an accuracy which is comparable to that of the best model (model with lowest error or value)

Technique 1 - k-Fold Cross Validation

To find the best choice of lambda one could use k-fold Cross validation. The procedure goes as follows (James et al., 2013):

1. Randomly divide all observations into k subsets or folds of approximately the same size.
2. For $m = 1, 2, \ldots k$ and for a given choice of lambda, we train our model on observations not lying in the m_{th} fold (use the remaining $k-1$ folds) and validate it on the m_{th} fold, each time recording the mean square error where $MSE^{\lambda} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
3. Given the k estimates of the MSE (test error), we compute the average k-fold CV error over all folds with the following formula:

$$CV(\lambda) = \frac{1}{k}\sum_{i=1}^{n} MSE_i^{\lambda}$$

4. This process is repeated for a range of values of lambda $\lambda = \lambda_1, \lambda_2, ...\lambda_z$ for a choice of z, allowing us to obtain $CV(\lambda_1), CV(\lambda_2), ...CV(\lambda_z)$ which can then be plotted and compared.
5. Obtain λ_{opt}^{b} using the Best λ approach where $\lambda_{opt}^{b} = \underset{\lambda \in \{\lambda_1, ...\lambda_z\}}{\text{argmin}} CV(\lambda)$. Obtain λ_{opt}^{se} using One Standard Error Rule approach.

Technique 2 - AIC

AIC or more explicitly Akaike Information Criterion is defined as (James et al., 2013):

$$AIC = 2k - 2ln(\hat{L})$$

with k = number of parameters and $\hat{L}$ = maximum value of the likelihood function for the model, is a measure of a model's goodness of fit, penalised by the complexity of the model. When comparing different models, the model with the lowest AIC is preferred. Therefore, given a range of λs, one can evaluate the AIC of the model for each specific lambda, to find the best $\lambda = \lambda_{opt}^{b}$ (λ which minimises the AIC) and the one standard error rule $\lambda = \lambda_{opt}^{se}$.

Technique 3 - BIC

BIC or Bayesian Information criterion can be defined as(James et al., 2013):

$$BIC = ln(n)k - 2ln(\hat{L})$$

Where n is the number of observations, k is the number of parameters, and $\hat{L}$ is the maximized value of the log likelihood function. As with the AIC, when comparing different models, the lower the BIC the better. As a result the optimal tuning parameter can once again be found by finding the $\lambda = \lambda^{b}_{opt}$ which minimises the BIC or the $\lambda = \lambda^{se}_{opt}$ whose BIC value lies within one standard error of the lowest BIC.

2.4.2 Graphical Lasso - Finding λ_{opt}

When considering the choice of the optimal tuning parameter for the Graphical Lasso approach, all the techniques described above were considered. The techniques therefore used are the following:

1. k-fold Cross Validation - One standard error rule and Best λ approaches
2. AIC - One standard error rule and Best λ approaches
3. BIC - One standard error rule and Best λ approaches

The process of choosing optimal λs is as described previously.

2.5 Evaluation of optimal models and comparison of approaches

2.5.1 Metrics for evaluating sample performance

In all approaches for determining the conditional dependency structure of the variables $X = (X_1, X_2, .., X_p)$ and for each choice of optimal tuning parameter, metrics were used to assess the performance of each model. These were Sensitivity, False Positive Rate, Precision, G-Mean (Geometric Mean) and F_1-Score, calculated through the use of a confusion matrix. In this case the confusion matrix was of the following form (Luque et al., 2019; Tharwat, 2020):

		True edge set		
		Positive	Negative	Total
Predicted edge set	Positive	TP	FP	$TP + FP$
	Negative	FN	TN	$FN + TN$
		$TP + FN$	$FP + TN$	N

Using the number of True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN), one can use the following formulas to obtain the value for each metric (Hossin and Sulaiman, 2015; Tharwat, 2020):

$$Sensitivity = \frac{TP}{TP + FN}$$

$$FPR = 1 - Specificity = \frac{FP}{TN + FP}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$G - Mean = \sqrt{Sensitivity * Specificity} = \sqrt{\frac{TP}{TP + FN} * \frac{TN}{FP + TN}}$$

$$F_1 - Score = \frac{TP}{TP + 0.5 * (FP + FN)}$$

The graphical model we are estimating will either be very sparse, very dense or or somewhere in between. The specific metrics were thus chosen and used, for one to be able to evaluate the performance of models whilst accounting for this varying balance and imbalance between the two classes.

2.5.2 ROC curve

Based on the aforementioned techniques of choosing the optimal λ for the Node-wise 1 and 2 and Graphical Lasso Approaches, a fine grid of λs was identified, as the λs which yield the best results. To now therefore compare the overall performance of Node-wise Lasso 1, Node-wise Lasso 2 and Graphical Lasso Approach of estimating the true edge set of our graphical model, this grid of λs was used to yield three ROC curves, one for each approach. The Area under the ROC (AUC) was estimated for each and box plots of the AUC were also used to compare the three approaches.

The ROC or Receiver operating characteristic curve is a plot of the True Positive Rate (TPR/ Sensitivity) against the False Positive Rate (FPR) (formulas as stated above) (Marzban, 2004; Hoo et al., 2017). Given a certain approach, for each $\lambda_i \in \{\lambda_1, \lambda_2, ...\lambda_z\}$ where grid of $\lambda s = \{\lambda_1, \lambda_2, ...\lambda_z\}$, the TPR_i and FPR_i are calculated. Plotting TPR_i and FPR_i for $i \in 1, 2, ...z$, yields the ROC curve. To now choose the best approach of the three, the AUC can be used. A model with $AUC = 0.5$ represented by a diagonal line in the ROC plot, corresponds to a model which predicts whether an edge exists or not at random. A model with $AUC = 1$, corresponds to a model which perfectly predicts the connectivity of the nodes and thus predicts an estimated edge set exactly equivalent to the true edge set. As a result, the approach which yields the highest AUC, corresponding to the model with an ROC curve closest to the top left corner, is preferred out of the three.

2.6 Simulations

In this section, the methods for tuning the parameter λ will be evaluated and compared. Furthermore, the node wise lasso and the graphical lasso will be tested in their accuracy

to recover the true edge set E, which is our main interest in this study. In the first part, the different methods used for tuning the parameter λ will be implemented for each of the approaches for recovering the true edge set E. In the second part, this approaches will be compared by using the "best" method, found in the first part, for tuning λ in different settings.

Let n be the sample size and p be the dimension or number of variables. For reasons of space and time limitation, in the first part only the setting when $n > p$ and the sparsity is chosen by having each entry of the matrix Θ to be 0 with probability 0.9 and 0.5 with probability 0.1, will be consider. In order to have a better visualization and a lower running complexity, n was chosen to be 500 and p, 10. However, when comparing the approaches using the "best" resulting method different cases will be consider by varying each of the sparsity and the parameters n and p. The procedures are replicated 50 times and the mean of the different metrics is recorded, along with their standard error.

2.6.1 Node Wise Lasso First Approach (Appendix B.1)

The first approach under consideration is the node wise lasso to estimate the true set E by the set $\hat{E}_1$, so both the coefficients β_{jl} and β_{lj} need to be non-zero for the variables X_j and X_l to be connected. This hard threshold condition will make the estimated set very sparse, specially after applying the one standard error rule, which will permit λ to be greater than the optimal one to have less complexity and better results for prediction. The network structures after running one simulation are presented in Figure 2.1 and Figure 2.2. In this case the true network is very sparse and hence this set $\hat{E}_1$ can be beneficial when estimating E, however this is not always the case as this is just one simulation. Also note how BIC has less edges, and hence less connected variables as this method tends to shrink the coefficients further than the other methods when selecting the optimal λ. Furthermore, the tuning of the AIC one standard error rule approach is quite close to the true network.

The final table presented in Table 2.1 reflects this last conclusion about the AIC one standard error rule method, as it has the second highest Sensitivity, after the optimal tuning λ AIC. However the FPR for the AIC optimal λ method is greater than for the AIC one standard error rule. This is summarised in the Geometric Mean as it considers both Sensitivity and the FPR, where the AIC one standard error rule method outperforms the rest of the approaches. Hence, it is able to estimate with a high precision the true dependence of the variables. Another remark that can be observed from the table is that many entries are NaN for the metric of Precision and F_1-score. This is due to the fact that when running the 50 simulations, in many cases the methods did not estimate any connected edge, and hence the True Positives and False Positives where both 0, and hence neither the Precision nor the F_1-score existed. The high probability of the true set being sparse and the way in which the estimated set was computed made these NaN occurrences more probable.

Finally, analyzing the box-plots presented in Figure 2.3 and Figure 2.4 it is interesting to see the high variability of the Sensitivity for the BIC method, this can be explained by the fact that BIC tends to shrink the coefficients further and this explains how the FPR has very low variability and the values are very close to 0. The CV when choosing the optimal λ perform in a similar manner to the AIC methods.

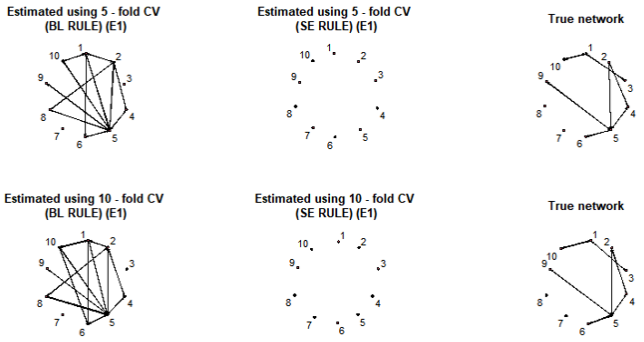

Figure 2.1: Estimated network for each of the variables after tuning λ by CV, applying the first approach of node wise lasso, where the existence of an edge between variables represent dependence between them

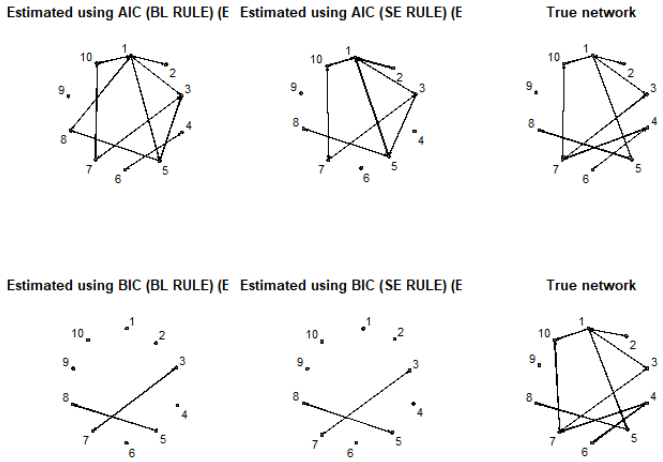

Figure 2.2: Estimated network for each of the variables after tuning λ by AIC and BIC, applying the first approach of node wise lasso, where the existence of an edge between variables represent dependence between them

Method	**Sensitivity**	FPR	**Precision**	**G-mean**	F_1**-Score**
CV (k=5, BL RULE)	0.789253968	0.095816900	NaN	0.817875909	NaN
CV (k=5, SE RULE)	0.025666667	0.000000000	NaN	0.054986175	NaN
CV (k=10, BL RULE)	0.831253968	0.097787361	0.488528250	0.848562313	0.615390995
CV (k=10, SE RULE)	0.002000000	0.000000000	NaN	0.006324555	NaN
AIC (BL RULE)	0.856920635	0.113761753	0.470816244	0.854605672	0.607729078
AIC (SE RULE)	0.844587302	0.095736142	0.510054770	0.857229055	0.636014178
BIC (BL RULE)	0.479873016	0.006924590	NaN	0.605130514	NaN
BIC (SE RULE)	0.368222222	0.005019288	NaN	0.494752829	NaN

Table 2.1: Means of the metrics after 50 simulations for each of the tuning λ methods in the node wise lasso first approach

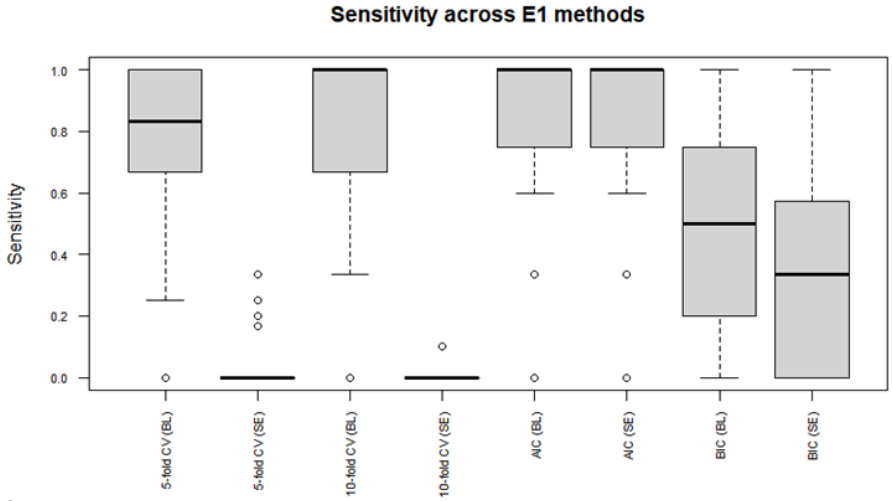

Figure 2.3: Box-plot of the Sensitivity for each of the tuning λ methods applying the first approach of node wise lasso

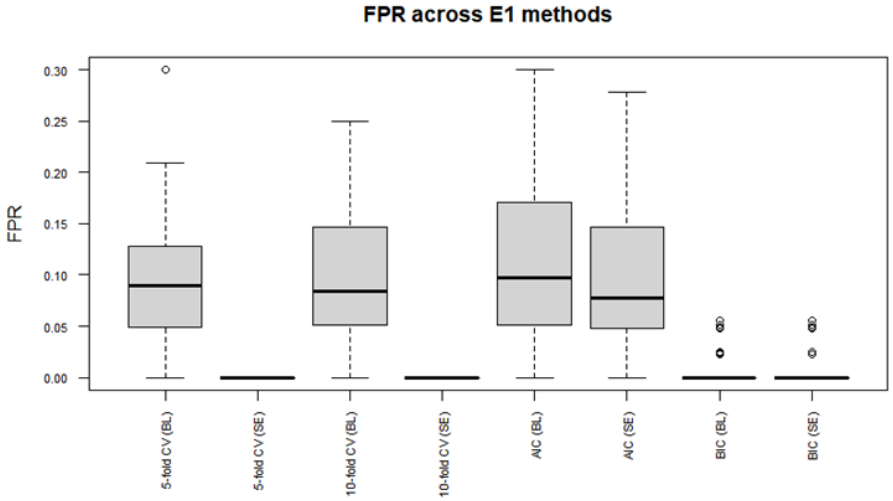

Figure 2.4: Box-plot of the FPR for each of the tuning λ methods applying the first approach of node wise lasso

2.6.2 Node Wise Lasso Second Approach (Appendix B.1)

In this second approach, node wise lasso will be implemented again, but this time the true set will be estimated by the set $\hat{E}_2$, so either one of the coefficients β_{jl} or β_{lj} need to be non-zero for the variables X_j and X_l to be connected. This relaxes the condition we had for $\hat{E}_1$, making the estimated network less sparse. The estimated network for one simulation are presented in Figure 2.5 and Figure 2.6. From these figures, it can be seen how if the true network is sparse, the methods of CV for selecting the optimal λ, as well as the AIC overestimate the number of edges in the network structure. However, the BIC, as it tends to shrink the coefficients further, does not seem to overestimate the number of edges in the graph and could improve its performance for estimating the true set E. Indeed this is the case, by looking at the metrics displayed in Table 2.2, as it increases its Sensitivity and Geometric Mean and lowers its FPR.

Nevertheless, it is worse than expected as one needs to be taken into account that this was just for one simulation and although the true network has higher probability of being sparse by default, it is not always the case. From Table 2.2, it can be seen that the AIC one standard error rule has the best performance metrics in summary, this is the same result as we had when estimating E by the node wise lasso first approach. It is surprising to observe how the precision is quite low in comparison with the other metrics even for the best performing methods. This can be due to the fact that even though many of the edges were correctly detected, there were many wrongly detected edges. One should also note that in this case the metrics are worse than when using the first node wise lasso approach as the methods for the second node wise lasso approach will tend to overestimate the number of edges.

Finally a summary of the Sensitivity and FPR is provided in Figure 2.7 and Figure 2.8 in the form of box-plots. The high variability of the Sensitivity for the BIC methods is again very present as well as its low variability for the FPR. The worst performing methods are clearly the CV standard error rule methods, as they shrink the coefficients too much, leaving few or no edges in the estimated graph. It can be concluded from this plot that the methods with highest Sensitivity and TPR are the CV choosing the best λ and the AIC methods.

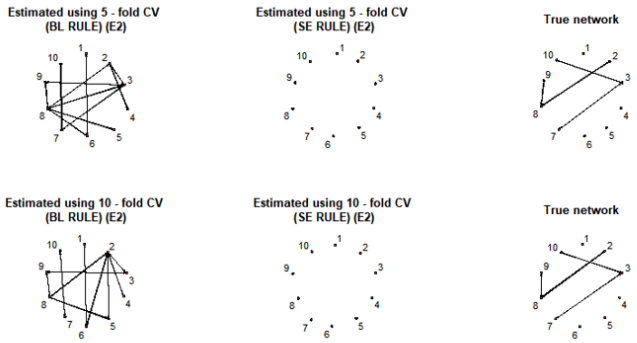

Figure 2.5: Estimated network for each of the variables after tuning λ by CV, applying the second approach of node wise lasso, where the existence of an edge between variables represent dependence between them

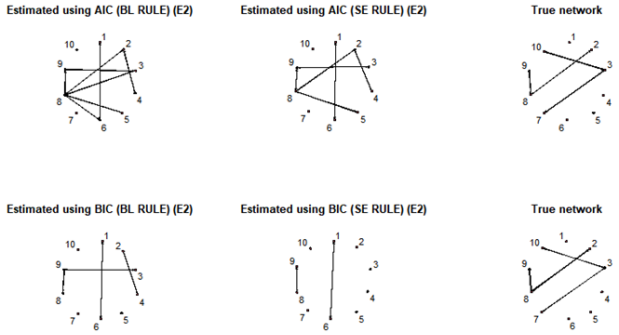

Figure 2.6: Estimated network for each of the variables after tuning λ by AIC and BIC, applying the second approach of node wise lasso, where the existence of an edge between variables represent dependence between them

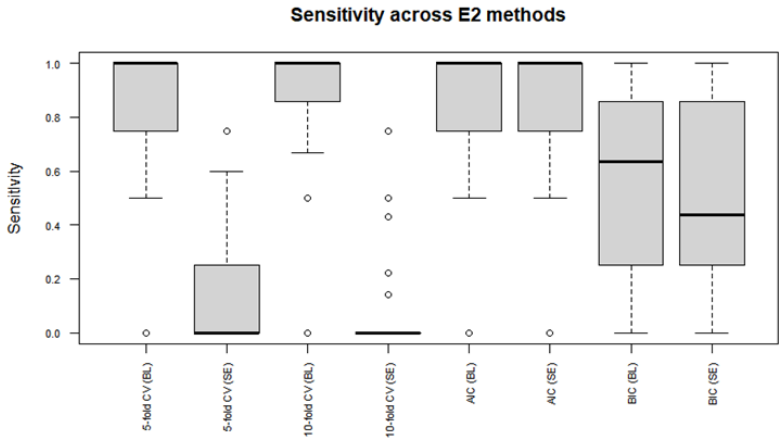

Figure 2.7: Box-plot of the Sensitivity for each of the tuning λ methods applying the second approach of node wise lasso

Method	**Sensitivity**	FPR	**Precision**	**G-mean**	F_1**-Score**
CV (k=5, BL RULE)	0.875198413	0.287965622	0.246782918	0.765246568	0.385004656
CV (k=5, SE RULE)	0.139100529	0.001626016	NaN	0.218893380	NaN
CV (k=10, BL RULE)	0.892460317	0.275860807	0.256531264	0.780741835	0.398512882
CV (k=10, SE RULE)	0.068121693	0.000000000	NaN	0.102571828	NaN
AIC (BL RULE)	0.871031746	0.268883031	0.265851204	0.774733883	0.4073679501
AIC (SE RULE)	0.852380952	0.239221881	0.284780336	0.783261912	0.4269250747
BIC (BL RULE)	0.561335979	0.056905512	NaN	0.648273801	NaN
BIC (SE RULE)	0.518346561	0.051095425	NaN	0.619406430	NaN

Table 2.2: Means of the metrics after 50 simulations for each of the tuning λ methods in the node wise lasso second approach

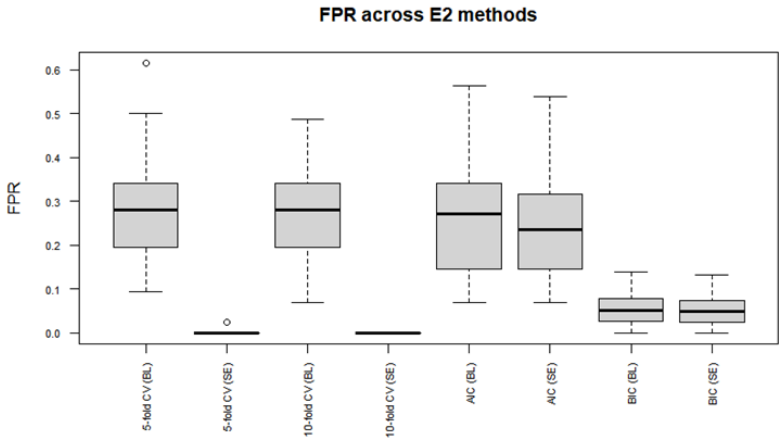

Figure 2.8: Box-plot of the FPR for each of the tuning λ methods applying the second approach of node wise lasso

2.6.3 Graphical Lasso (Appendix B.2)

The graphical lasso it is very different from the other node wise lasso approaches, in this case the set E is estimated by $\hat{E}_3$ in which two variables X_j and X_l are connected if $\hat{\Theta}_{jl} = 0$, where Θ is the inverse of the estimated covariance matrix.

The network structures after one simulation are displayed in Figure 2.9. The results described in the Table 2.3 for this approach tend to be worse than for the node wise lasso approaches. From having all NaN in the precision it can be observed that for all the tuning *lambda* methods there are many cases where no edges are detected and a graph containing only nodes is estimated. This indicates that the graphical lasso tends to select the optimal value of λ to be in the boundary, leading to more sparsity and to worst results, specially in the cases where the true network is dense enough. In this case the 5-fold CV seems to outperform the other methods as its Sensitivity is much higher, however it also has the highest FPR which indicates that although it is the method that detects correctly the highest number of connected variables, it is also the method which wrongly estimates the highest number of connected variables. Hence, other methods with a lower FPR will be preferred, even if that implies getting a smaller value of the Sensitivity metric.

Based on the Geometric Mean the best approach will be the AIC for optimal λ, as it is the one which balances best the Sensitivity and the FPR. The BIC methods perform poorly, this can be expected as if there are already many graphs with no edges detected by the other methods, then BIC will tend to shrink the coefficients even further. The methods of CV using the one standard error rule have improved compared to the node wise lasso approaches.

Box-plots of the Sensitivity and FPR for these approaches are displayed in Figure 2.10 and Figure 2.11, the variability of the Sensitivity for most of the methods is very high. This is not very comforting as the results are very variable, so we could apply the method and get the best result or the worst one. Furthermore, this variability also translates to the CV methods for the FPR metric and this error can be very large. However for the FPR for the AIC methods it is reassuring, as it achieves very low values close to zero and with low variability. The BIC method has an FPR close zero with no or very low variability. This is due to the fact that the Sensitivity is very close to zero, so it is not able to make mistakes in estimating connected edges as most of the times it does not even estimate any connected edges.

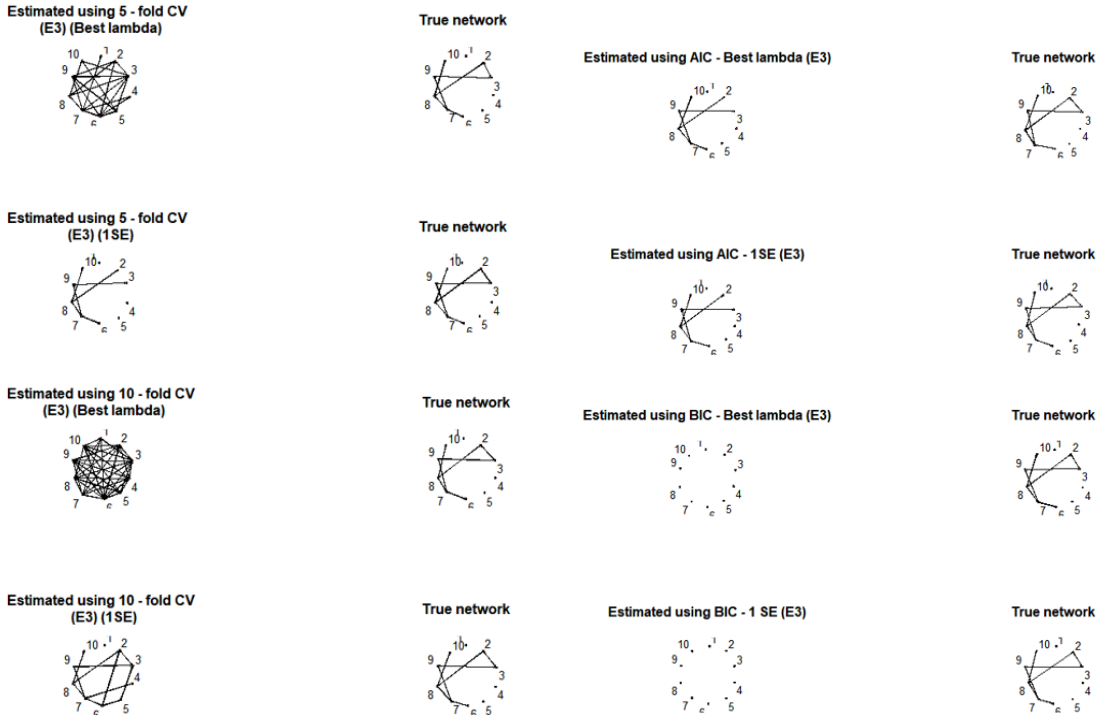

Figure 2.9: Estimated network for each of the variables after tuning λ by CV, AIC and BIC, applying the graphical lasso, where the existence of an edge between variables represent dependence between them

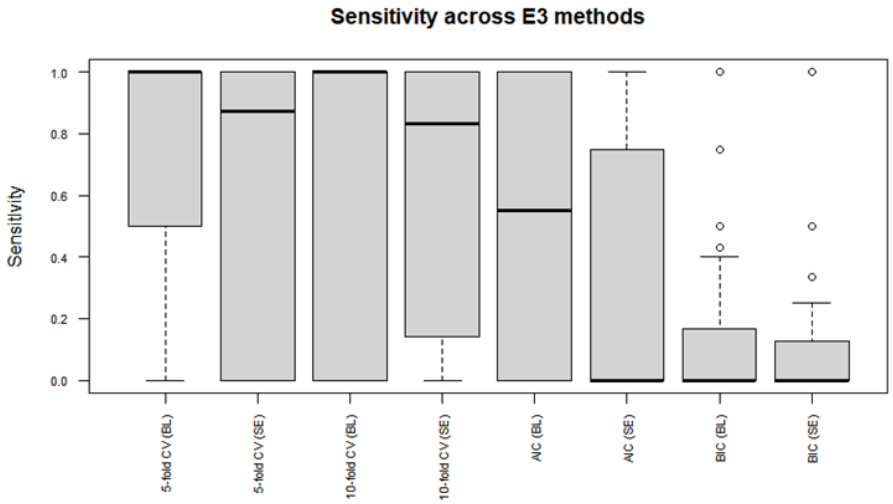

Figure 2.10: Box-plot of the Sensitivity for each of the tuning λ methods applying the graphical lasso

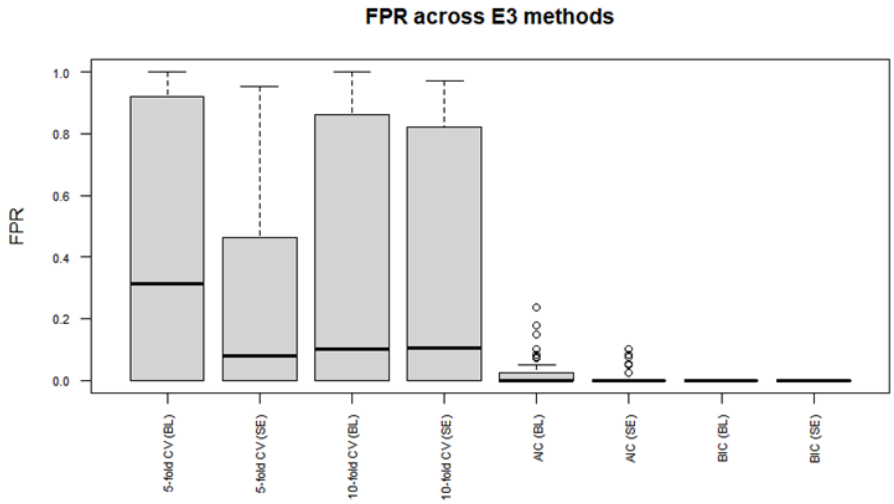

Figure 2.11: Box-plot of the FPR for each of the tuning λ methods applying the graphical lasso

Method	**Sensitivity**	FPR	**Precision**	**G-mean**	F_1**-Score**
CV (k=5, BL RULE)	0.750000000	0.427283208	NaN	0.448778045	NaN
CV (k=5, SE RULE)	0.624920635	0.246014743	NaN	0.506350062	NaN
CV (k=10, BL RULE)	0.615444444	0.347828825	NaN	0.389783813	NaN
CV (k=10, SE RULE)	0.614619048	0.343032023	NaN	0.4324596488	NaN
AIC (BL RULE)	0.538031746	0.029053565	NaN	0.598232583	NaN
AIC (SE RULE)	0.307031746	0.009417843	NaN	0.346416946	NaN
BIC (BL RULE)	0.117404762	0.000000000	NaN	0.176365932	NaN
BIC (SE RULE)	0.077690476	0.000000000	NaN	0.139806801	NaN

Table 2.3: Means of the metrics after 50 simulations for each of the tuning λ methods in the graphical lasso

2.6.4 Comparison of the approaches using the "best" method (Appendix B.3)

After having compared the different tuning λ methods for each of the approaches for estimating the true set E, these estimation methods will be compared by applying a specific tuning λ technique (for consistency) that will be decided among the ones considered in the above sections. Many techniques can be chosen to be the "best" one, as the CV had a good performance for most of the methods, however its FPR was also higher, especially for the graphical lasso approach. These results are also very dependent on the chosen Θ and X, hence it is hard to choose the appropriate one. For the node wise lasso approaches the best method was the AIC one standard error rule, it had a very high Sensitivity and a very low FPR, and even though for the graphical lasso the Sensitivity was not as good as other methods, it had a very low FPR close to zero. Furthermore in other settings when the sparsity is not very high it tends to outperform the other methods. Hence, for these reasons and to be able to experiment with other tuning methods rather than the classical CV, the AIC one standard error rule will be considered.

There will be six cases under consideration, this will result from changing the parameters n,p and the sparsity level. The first three cases will focus on the common case when $n > p$, where n is chosen to be 500 and p is chosen to be 10. These parameters are chosen to account for the running complexity and for visualization purposes. In the first case, the sparsity is chosen by having each entry of the matrix Θ to be 0 with probability 0.9 and 0.5 with probability 0.1. For the second case, the sparsity is chosen by having each entry of the matrix Θ to be 0 and 0.5 both with probability 0.5. In the third case, the sparsity is chosen by having each entry of the matrix Θ to be 0 with probability 0.1 and 0.5 with probability 0.9. Hence the first case will tend to have high sparsity, the second case medium sparsity and the last case low sparsity.

For the last three cases the sparsity considered will be the same as for the first cases, however now the case of $n < p$ will be considered. In these cases to have an efficient running complexity and for purposes of visualization p is chosen to be 10 and n is chosen to be 5. For each of the cases the procedure is done 50 times and the means of the relevant metrics along with its standard error are recorded. Furthermore the ROC curves are displayed and its mean area is calculated for each of the node wise and graphical lasso methods. The code for formulating the ROC curves can be found in (Appendix B.4)

Case 1

In this case $n > p$ and high sparsity is under consideration, which is the same case considered in the above sections. As a first analysis the ROC curves (Figure 2.13) and the Area Under the Curve (AUC) are considered for the two node wise lasso and graphical lasso approaches. From Figure 2.13, it can be observed that the mean AUC is very similar for all approaches, with the second approach of the node wise lasso achieving the highest and also less variability. The standard error of the means of the metrics is displayed in Table 2.5, however it is very low for all the metrics for every method. The approach that obtains the best balance between Sensitivity and FPR is the node wise lasso first approach and the worst is the graphical lasso for estimating E.

The box-plots of the metrics of Sensitivity, FPR and G-Mean are presented in Figure 2.14, Figure 2.15 and Figure 2.16, respectively. The high variability of the G-Mean for the graphical lasso is something to take into account, however this was our worst performing method for recovering E, hence not likely to be implemented in this case. The network structure for one simulation displayed in Figure 2.12 gives a good representation of what is going on behind implementing these methods, the second approach of the node wise lasso tends to overestimate the number of edges of the true model, while the graphical lasso tends to overestimate this number. The first approach of the node wise lasso is the one that comes closer to the true set E.

Method	**Sensitivity**	FPR	**Precision**	**G-mean**	F_1**-Score**
AIC (SE RULE) (E1)-MEAN	0.84661905	0.09648983	0.48380314	0.84964240	0.61573998
AIC (SE RULE) (E2)-MEAN	0.93080952	0.23936379	0.28360895	0.83645021	0.41933470
AIC (SE RULE) (E3) - MEAN	0.28185714	0.01523090	NaN	0.34893651	NaN

Table 2.4: Means of the metrics after 50 simulations ($n > p$, high sparsity) for the AIC, one standard error rule tuning λ for the node wise lasso and the graphical lasso

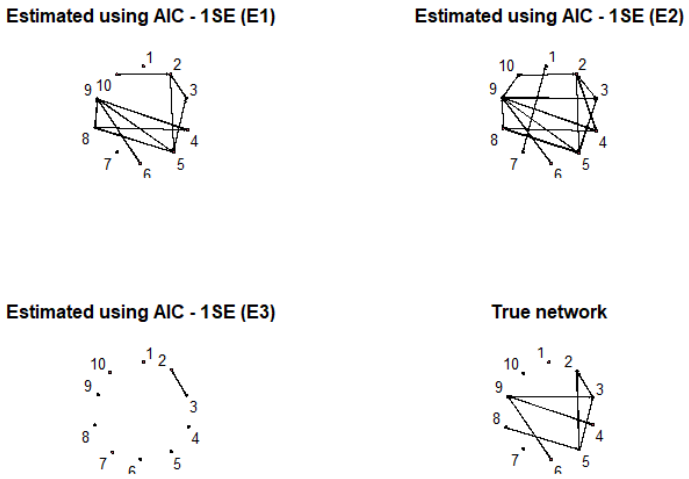

Figure 2.12: Estimated network for each of the approaches for Case 1 ($n > p$, high sparsity) and the true network, where the existence of an edge between variables represent dependence between them

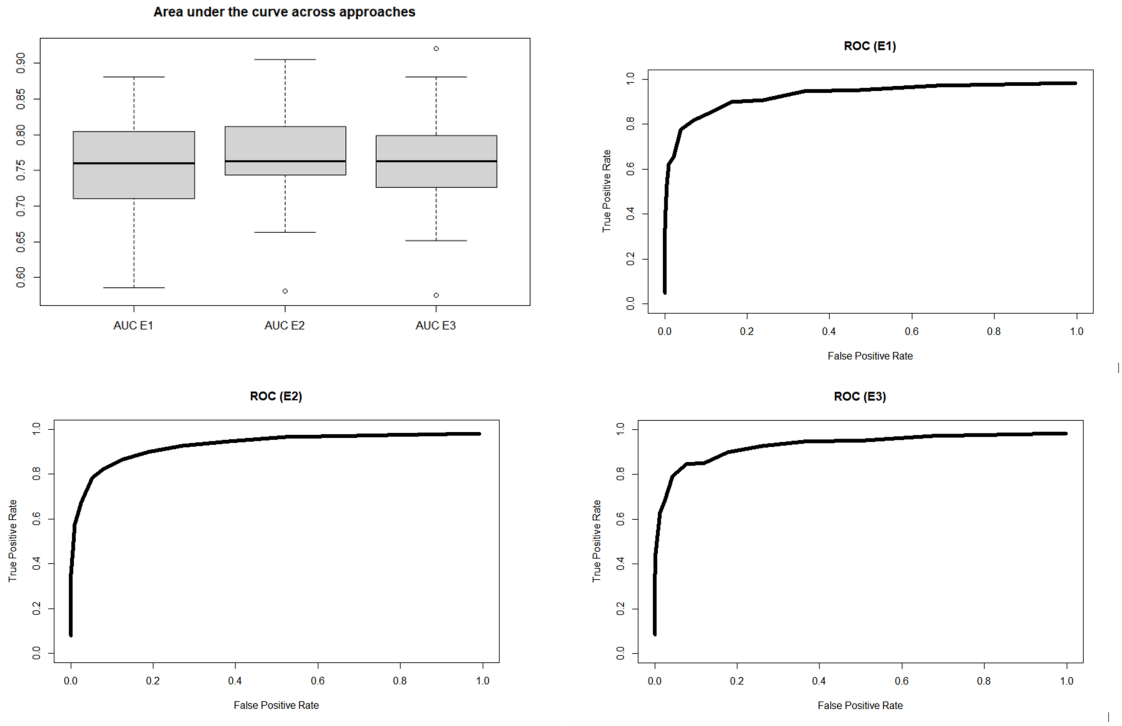

Figure 2.13: ROC and AUROC plots for each of the approaches for Case 1 ($n > p$, high sparsity)

Method	Sensitivity	FPR	Precision	G-mean	F_1-Score
AIC (SE RULE) (E1) - SE	0.004882592	0.001439823	0.004251341	0.004054466	0.004545153
AIC (SE RULE) (E2) - SE	0.002725021	0.002197354	0.002596471	0.001847256	0.003092497
AIC (SE RULE) (E3) - SE	0.007092742	0.001207547	NaN	0.007729215	NaN

Table 2.5: Standard error of the means of the metrics after 50 simulations ($n > p$, high sparsity) for the AIC, one standard error rule tuning λ for the node wise lasso and the graphical lasso

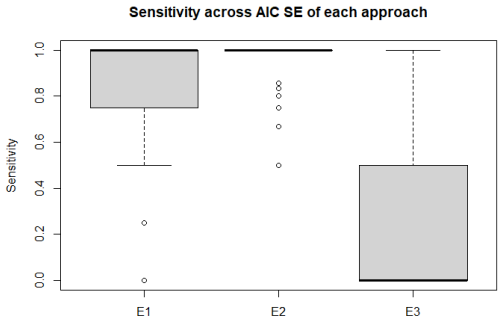

Figure 2.14: Box-plot of the Sensitivity for each of the approaches for Case 1 ($n > p$, high sparsity)

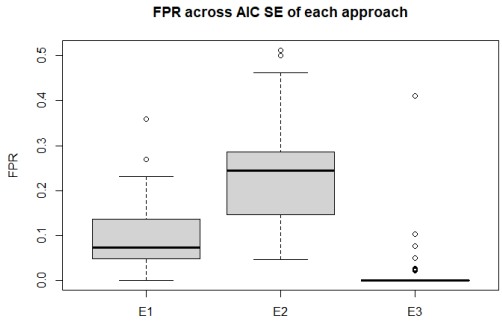

Figure 2.15: Box-plot of the FPR for each of the approaches for Case 1 ($n > p$, high sparsity)

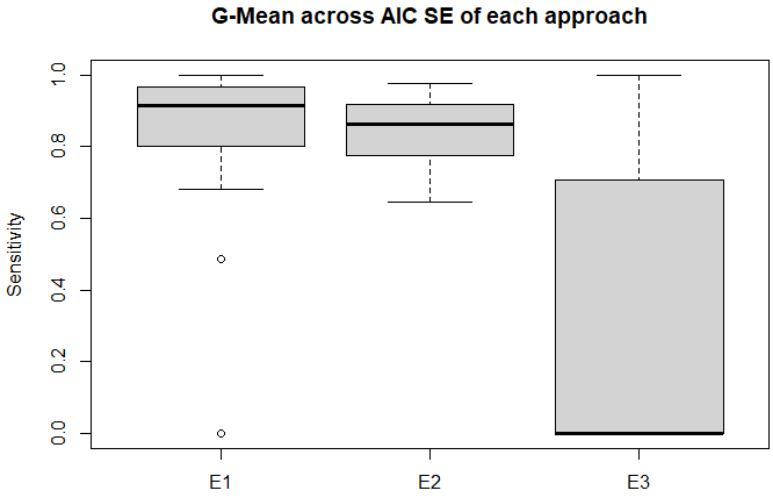

Figure 2.16: Box-plot of the $G - Mean$ for each of the approaches for Case 1 ($n > p$, high sparsity)

Case 2

This case will be similar to the above but now the probability of the existence of edges between the variables will be set to 0.5. Therefore the set E will tend to have medium sparsity. The network structure for one simulation is displayed in Figure 2.17 and the ROC curves in Figure 2.18. The results for this case are improved with respect to Case 1 for all the metrics as can be seen in Table 2.6, Figure 2.19, Figure 2.20 and Figure 2.21. The method with the highest balance between the Sensitivity and FPR is the first node wise lasso approach, however the others do not differ significantly. Similarly to Case 1, the standard error is very close to 0 for all the methods, which indicates that the mean of the different metrics have low variability, this is displayed in Table 2.7. The only alarming result to consider after this excellent result is the outliers for the Sensitivity of the graphical lasso presented in the box-plot in Figure 2.19, especially since they are very close to zero. This indicates that for some of the simulations the worst possible result for Sensitivity is obtained, and although there are just a few isolated cases they have to be taken into account in further analysis when applying these methods.

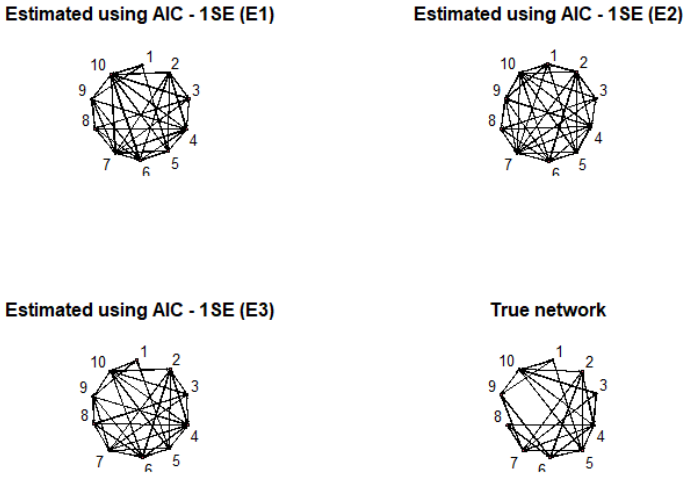

Figure 2.17: Estimated network for each of the approaches for Case 2 ($n > p$, medium sparsity) and the true network, where the existence of an edge between variables represent dependence between them

Method	Sensitivity	FPR	Precision	G-mean	F_1-Score
AIC (SE RULE) (E1)-MEAN	0.9564040	0.3440140	0.7450003	0.7872335	0.8342751
AIC (SE RULE) (E2)-MEAN	0.9815272	0.4988458	0.6694085	0.6961796	0.7942819
AIC (SE RULE) (E3) - MEAN	0.8877398	0.3464847	0.7537753	0.7352520	0.7846968

Table 2.6: Means of the metrics after 50 simulations ($n > p$, medium sparsity) for the AIC, one standard error rule tuning λ for the node wise lasso and the graphical lasso

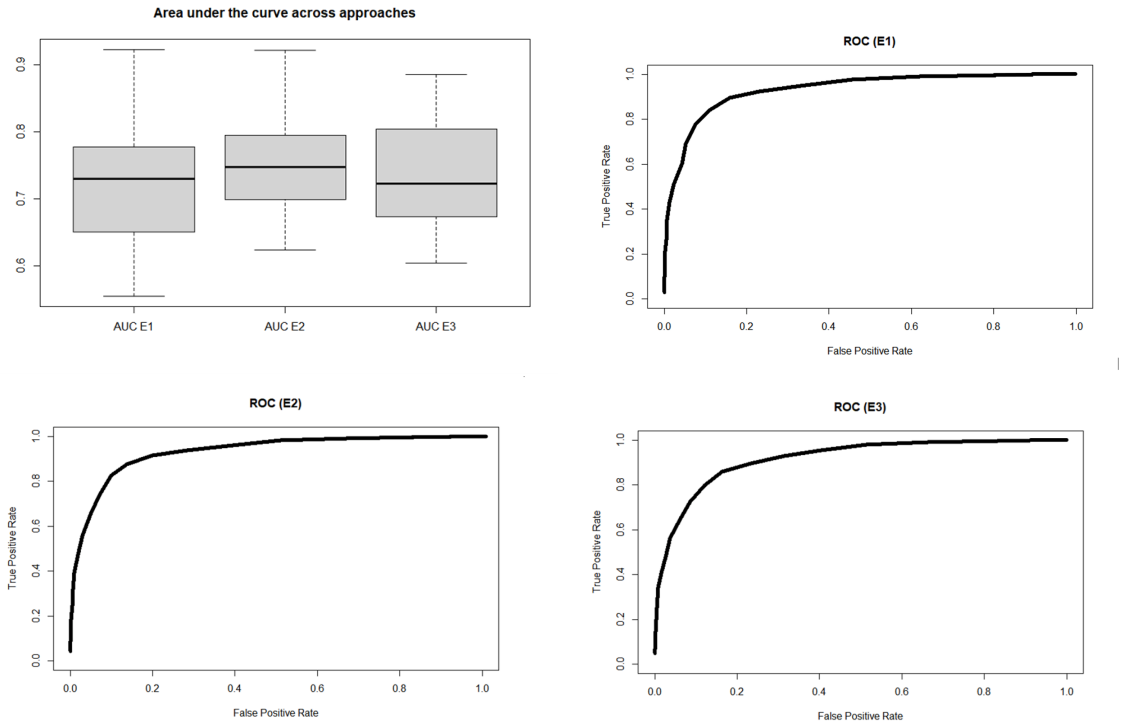

Figure 2.18: ROC and AUROC plots for each of the approaches for Case 2 ($n > p$, medium sparsity)

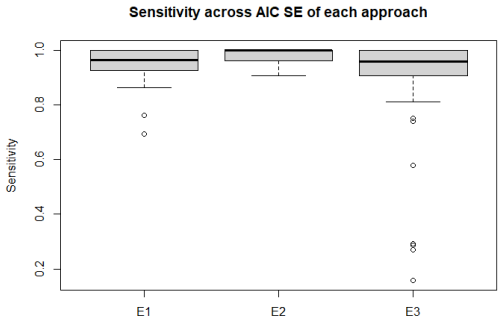

Figure 2.19: Box-plot of the Sensitivity for each of the approaches for Case 2 ($n > p$, medium sparsity)

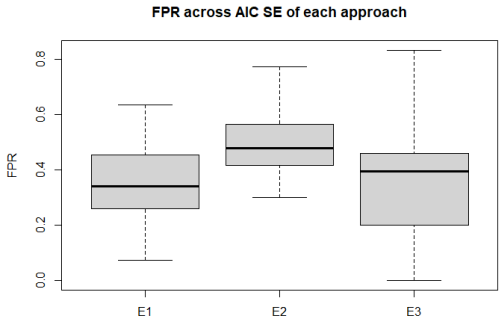

Figure 2.20: Box-plot of the FPR for each of the approaches for Case 2 ($n > p$, medium sparsity)

Method	Sensitivity	FPR	Precision	G-mean	F_1-Score
AIC (SE RULE) (E1) - SE	0.0012253647	0.0025985418	0.0015803440	0.0015911741	0.0010937369
AIC (SE RULE) (E2) - SE	0.0006164373	0.0024021616	0.0013184661	0.0018161437	0.0009980099
AIC (SE RULE) (E3) - SE	0.0041430911	0.0038326921	0.0021975577	0.0023972306	0.0025513377

Table 2.7: Standard error of the means of the metrics after 50 simulations ($n > p$, medium sparsity) for the AIC, one standard error rule tuning λ for the node wise lasso and the graphical lasso

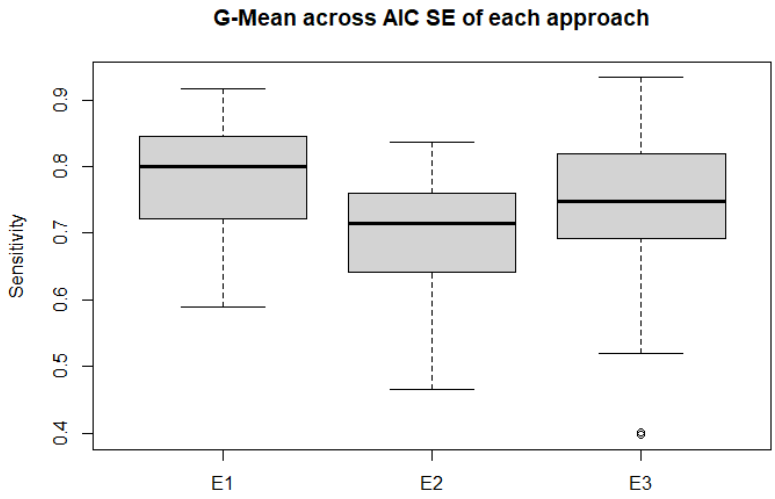

Figure 2.21: Box-plot of the $G - Mean$ for each of the approaches for Case 2 ($n > p$, medium sparsity)

Case 3

In this case $n > p$ and low sparsity is under consideration. Here the sparsity is chosen by having each entry of the matrix Θ to be 0 with probability 0.1 and 0.5 with probability 0.9. The network structure for one simulation is presented in Figure 2.22. The metric results displayed in Table 2.8 and Table 2.9, are very similar to the ones of Case 2, however the FPR is very high for all of the methods, which implies that the methods tend to overestimate the number of connected edges between the variables and makes the G-Mean score very low. This also can be seen in the box-plots of FPR and G-Mean in Figure 2.25 and Figure 2.26. Similarly to Case 1, the existence of very low outliers for Sensitivity still remains as can be seen in Figure 2.24.

Method	Sensitivity	FPR	Precision	G-mean	F_1-Score
AIC (SE RULE) (E1)-MEAN	0.9628328	0.6712222	0.9292994	0.4861885	0.9453026
AIC (SE RULE) (E2)-MEAN	0.9829805	0.7651587	0.9196487	0.3948481	0.9497941
AIC (SE RULE) (E3) - MEAN	0.8686147	0.7042222	NaN	0.3024887	NaN

Table 2.8: Means of the metrics after 50 simulations ($n > p$, low sparsity) for the AIC, one standard error rule tuning λ for the node wise lasso and the graphical lasso

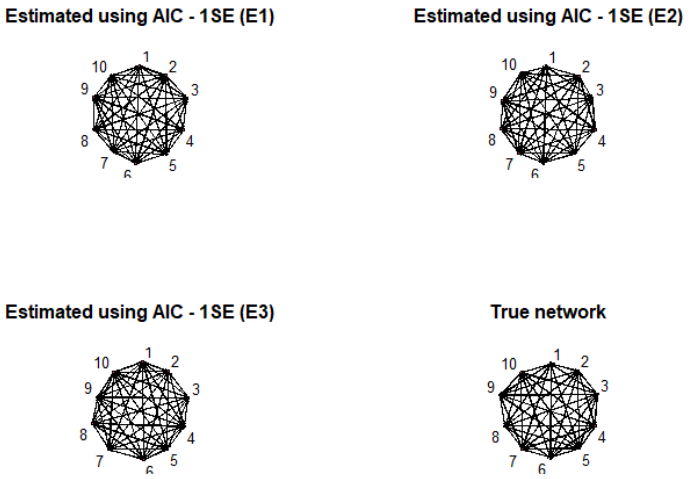

Figure 2.22: Estimated network for each of the approaches for Case 3 ($n > p$, low sparsity) and the true network, where the existence of an edge between variables represent dependence between them

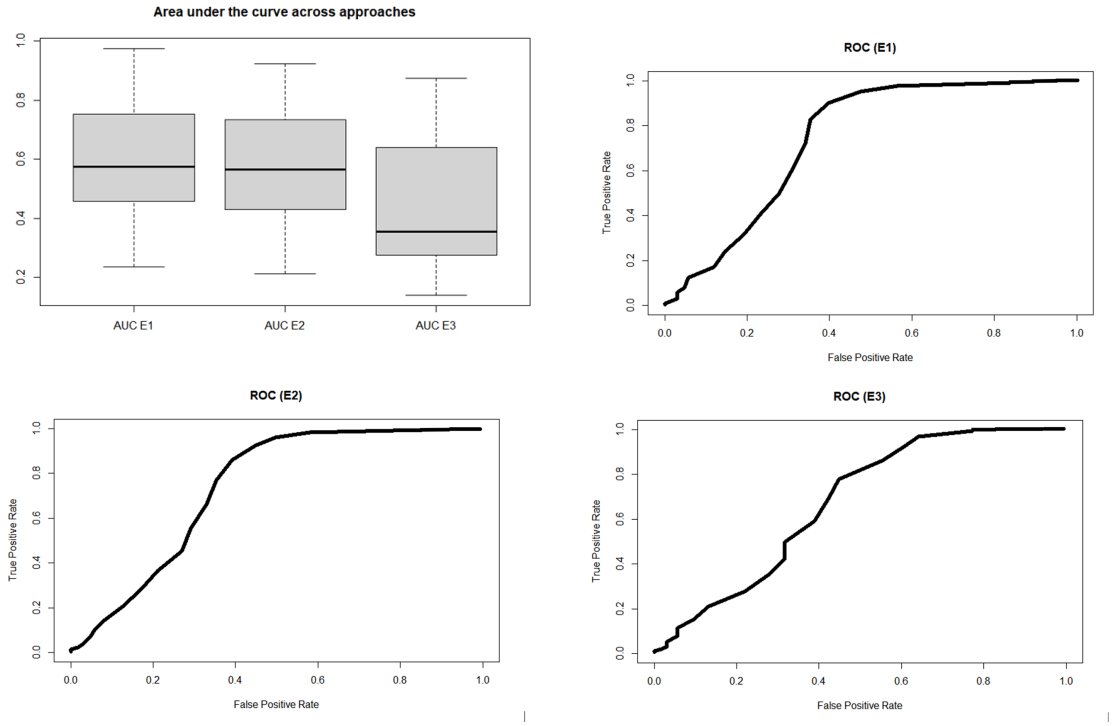

Figure 2.23: ROC and AUROC plots for each of the approaches for Case 3 ($n > p$, low sparsity)

Method	Sensitivity	FPR	Precision	G-mean	F_1-Score
AIC (SE RULE) (E1) - SE	0.0008090848	0.0048258633	0.0007599866	0.0057858460	0.0006666603
AIC (SE RULE) (E2) - SE	0.0005001823	0.0040142181	0.0008082499	0.0055701618	0.0005440550
AIC (SE RULE) (E3) - SE	0.0060544373	0.0067671788	NaN	0.0060195915	NaN

Table 2.9: Standard error of the means of the metrics after 50 simulations ($n > p$, low sparsity) for the AIC, one standard error rule tuning λ for the node wise lasso and the graphical lasso

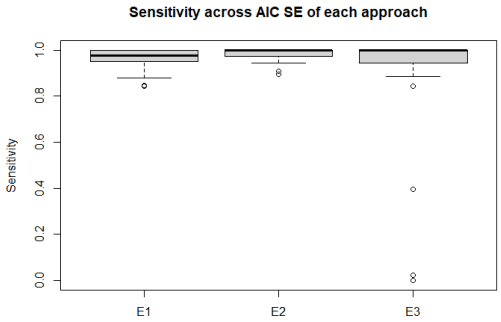

Figure 2.24: Box-plot of the Sensitivity for each of the approaches for Case 3 ($n > p$, low sparsity)

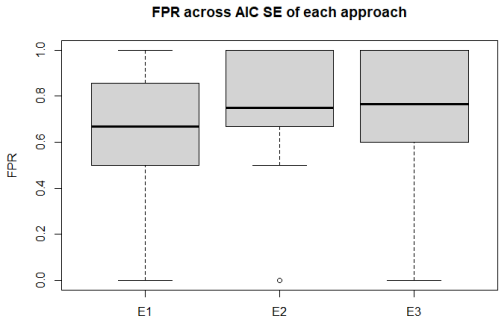

Figure 2.25: Box-plot of the FPR for each of the approaches for Case 3 ($n > p$, low sparsity)

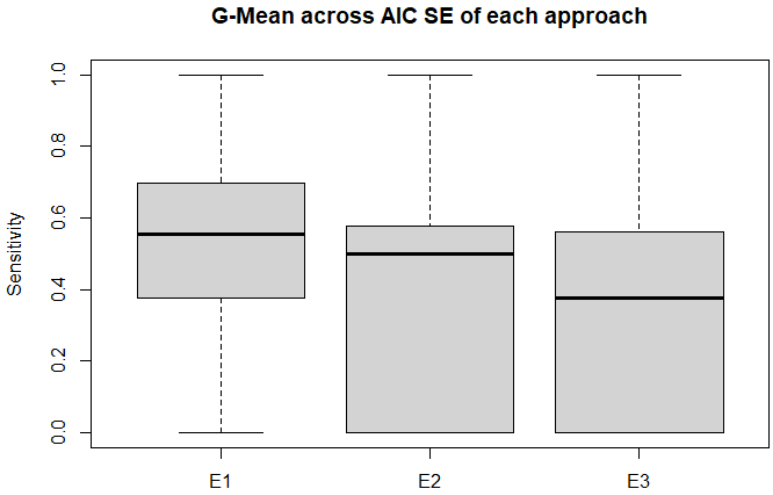

Figure 2.26: Box-plot of the $G - Mean$ for each of the approaches for Case 3 ($n > p$, low sparsity)

Case 4

For the following three cases $n < p$ will be considered. The case of $n = p$ could also have been under study, however it provides similar results to the $n < p$ case, a bit better performance, and hence it will be more interesting to consider the most extreme case for the parameters n and p. The *ROC* curves are omitted from being displayed in the report due to the terrible performance of these methods in detecting the true set E, performing in many cases worse than a random classifier. Therefore they are not of our interest. In this Case 4, the sparsity is chosen by having each entry of the matrix Θ to be 0 with probability 0.9 and 0.5 with probability 0.1, leading to cases of high sparsity.

One of the reasons for the general Lasso approach to be introduced was to lower the multicollinearity between the variables and the variance of a model in high dimensional settings, by shrinking some of the coefficients towards zero and hence performing variable selection (James et al., 2013). In our case, the problem considered is trying to estimate the dependence between the variables by regressing them against all the other variables and shrinking some coefficients to zero in the case of the node wise lasso approaches (James et al., 2013); and by estimating the inverse co-variance matrix Θ for data X in the case of the graphical lasso and letting some of these entries shrink to zero (Friedman et al., 2010). In this case as $n < p$, the approaches will give the penalty term a higher weight and hence we will be expecting further shrinkage and sparse estimated sets.

This can be seen in Figure 2.27 where the network structure is displayed for one of the simulations. The means of the metrics, presented in Table 2.10, are terrible in this case, having Sensitivities very low which can be due to the fact that as further shrinkage is implemented, for many of the cases the methods will not even estimate a single edge and will just be node graphs without edges, as the NaN entries in the Precision indicate. The standard errors of the means of the metrics, displayed in Figure 2.11 are very low and close to zero which implies that the mean reflects reality with a high accuracy and confirms the poor performance of the methods. A very interesting result obtained from the box-plots (Figure 2.28, Figure 2.29 and Figure 2.30), is the outliers obtained in the Sensitivity and FPR box-plots for the graphical lasso method, as in some cases it is able to obtain a Sensitivity higher than 0.5, even reaching 1 in one case. These outliers will be important to consider in some further analysis.

Method	**Sensitivity**	FPR	**Precision**	**G-mean**	F_1**-Score**
AIC (SE RULE) (E1)-MEAN	0.01066667	0.03653427	NaN	0.02484422	NaN
AIC (SE RULE) (E2)-MEAN	0.10923810	0.13783750	0.08131047	0.18379889	0.093227687
AIC (SE RULE) (E3) - MEAN	0.13600000	0.14843640	NaN	0.09425185	NaN

Table 2.10: Means of the metrics after 50 simulations ($n < p$, high sparsity) for the AIC, one standard error rule tuning λ for the node wise lasso and the graphical lasso

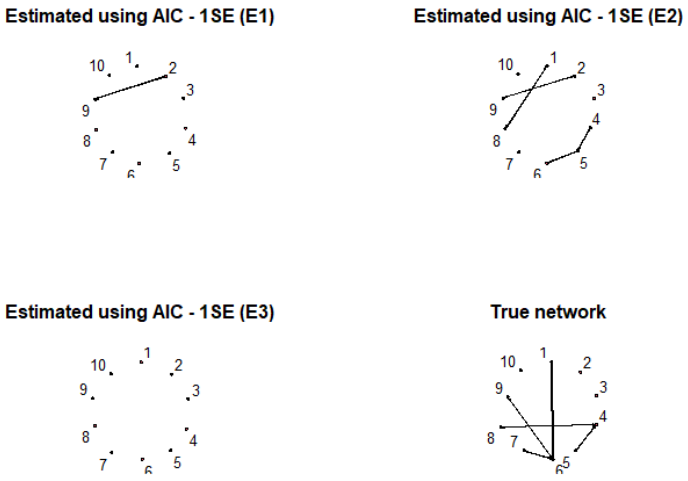

Figure 2.27: Estimated network for each of the approaches for Case 4 ($n < p$, high sparsity) and the true network, where the existence of an edge between variables represent dependence between them

Method	Sensitivity	FPR	Precision	G-mean	F_1-Score
AIC (SE RULE) (E1) - SE	0.0008565077	0.0005111811	NaN	0.0019890538	NaN
AIC (SE RULE) (E2) - SE	0.0033737058	0.0011088066	0.0026070891	0.0049491485	0.0029412651
AIC (SE RULE) (E3) - SE	0.0055247782	0.0057696972	NaN	0.0036387156	NaN

Table 2.11: Standard error of the means of the metrics after 50 simulations ($n < p$, high sparsity) for the AIC, one standard error rule tuning λ for the node wise lasso and the graphical lasso

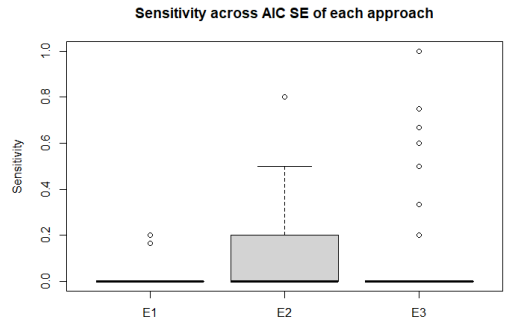

Figure 2.28: Box-plot of the Sensitivity for each of the approaches for Case 4 ($n < p$, high sparsity)

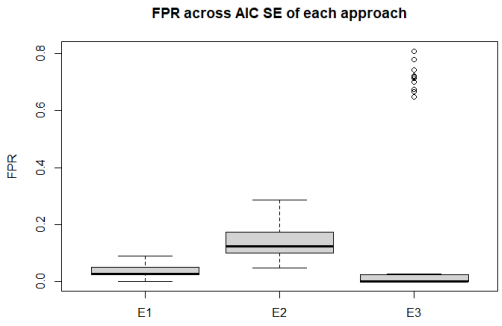

Figure 2.29: Box-plot of the FPR for each of the approaches for Case 4 ($n < p$, high sparsity)

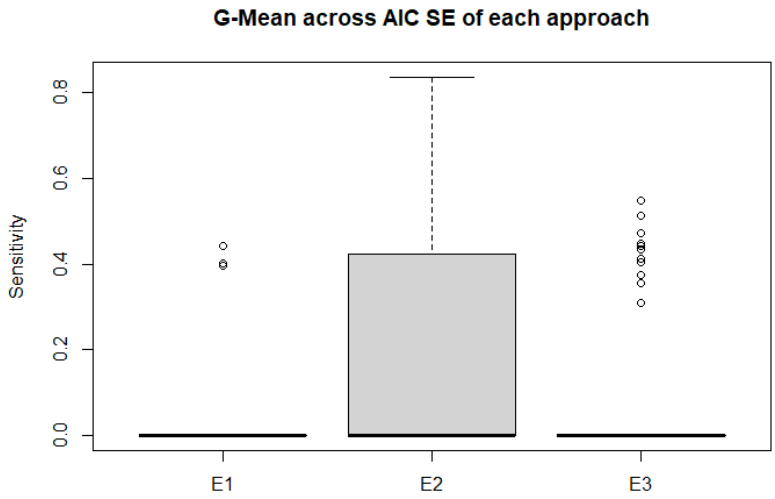

Figure 2.30: Box-plot of the $G-Mean$ for each of the approaches for Case 4 ($n < p$, high sparsity)

Case 5

This case will be similar to the above but now the probability of the existence of edges between the variables will be set to 0.5. Therefore the set E will tend to have medium sparsity. The network structure is presented in Figure 2.31. Similar results as for the previous case will be obtained for this setting, or even worse as now the true network is less sparse and the methods will still tend to shrink the coefficients further as for the above case. These results are displayed in Table 2.12, Table 2.13, Figure 2.32, Figure 2.33 and Figure 2.34. Once again, it needs to be stated that there are many outliers appearing for the Sensitivity and FPR for the graphical lasso that are interesting and need to be further analyzed.

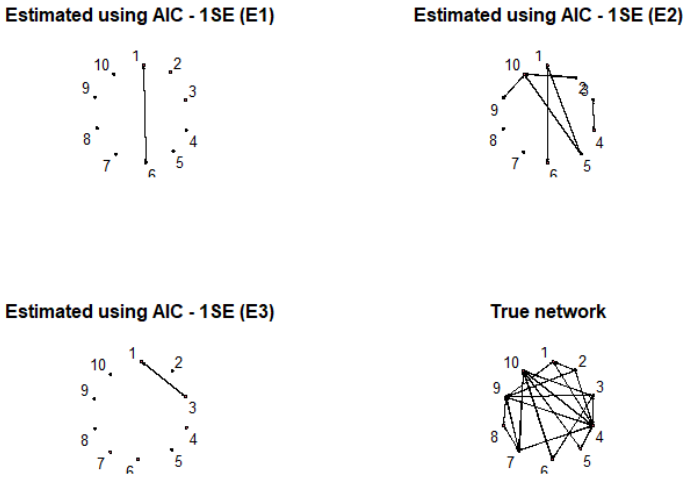

Figure 2.31: Estimated network for each of the approaches for Case 5 ($n < p$, medium sparsity) and the true network, where the existence of an edge between variables represent dependence between them

Method	Sensitivity	FPR	Precision	G-mean	F_1-Score
AIC (SE RULE) (E1)-MEAN	0.03242356	0.04107093	NaN	0.12944978	NaN
AIC (SE RULE) (E2)-MEAN	0.14606707	0.15233738	0.47212554	0.33217450	0.22310844
AIC (SE RULE) (E3) - MEAN	0.05676303	0.06361018	NaN	0.05016321	NaN

Table 2.12: Means of the metrics after 50 simulations ($n < p$, medium sparsity) for the AIC, one standard error rule tuning λ for the node wise lasso and the graphical lasso

Method	Sensitivity	FPR	Precision	G-mean	F_1-Score
AIC (SE RULE) (E1) - SE	0.0007157559	0.0006380114	NaN	0.0024324936	NaN
AIC (SE RULE) (E2) - SE	0.0016366025	0.0013458333	0.0040988259	0.0023313468	0.0023391971
AIC (SE RULE) (E3) - SE	0.0037216964	0.0037042498	NaN	0.0027156560	NaN

Table 2.13: Standard error of the means of the metrics after 50 simulations ($n < p$, medium sparsity) for the AIC, one standard error rule tuning λ for the node wise lasso and the graphical lasso

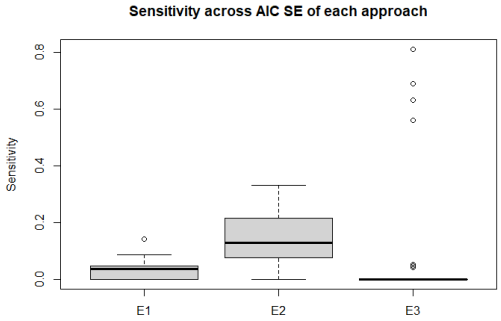

Figure 2.32: Box-plot of the Sensitivity for each of the approaches for Case 5 ($n < p$, medium sparsity)

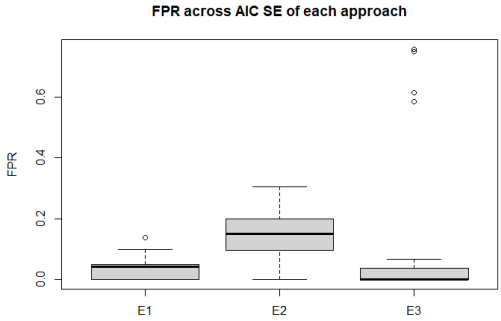

Figure 2.33: Box-plot of the FPR for each of the approaches for Case 5 ($n < p$, medium sparsity)

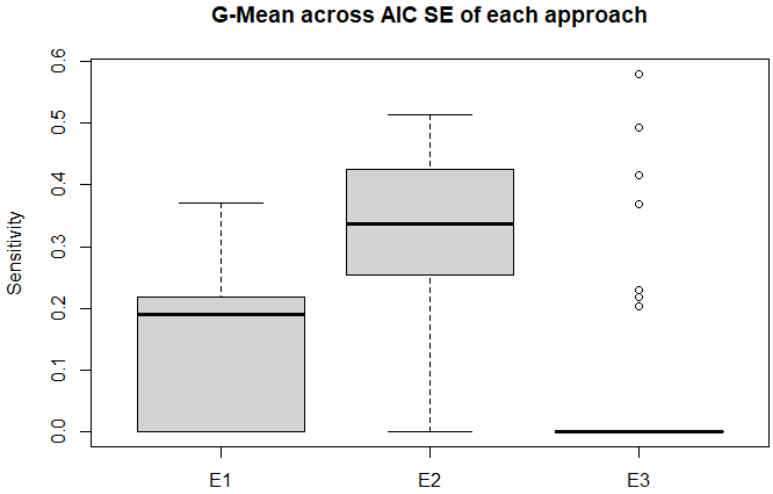

Figure 2.34: Box-plot of the $G - Mean$ for each of the approaches for Case 5 ($n < p$, medium sparsity)

Case 6

In this case $n < p$ and low sparsity is under consideration. Here the sparsity is chosen by having each entry of the matrix Θ to be 0 with probability 0.1 and 0.5 with probability 0.9. The results are very similar to the ones of the two above cases considered, however in this setting the true network tends to be very dense and hence the expected results for the metrics should be worse than for the other cases considered above. However, surprisingly this is not always the case, but similar metrics are obtained. The network structure is displayed in Figure 2.35, while the results are presented in Table 2.14, Table 2.15, Figure 2.36, Figure 2.37 and Figure 2.38.

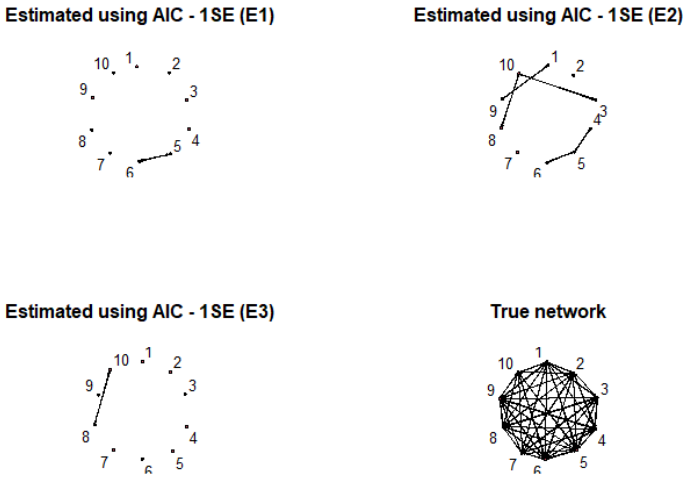

Figure 2.35: Estimated network for each of the approaches for Case 6 ($n < p$, low sparsity) and the true network, where the existence of an edge between variables represent dependence between them

Method	Sensitivity	FPR	Precision	G-mean	F_1-Score
AIC (SE RULE) (E1)-MEAN	0.02783122	0.03904762	NaN	0.12666341	NaN
AIC (SE RULE) (E2)-MEAN	0.12165145	0.13140476	0.88941270	0.31668926	0.21223959
AIC (SE RULE) (E3) - MEAN	0.12668585	0.09946032	NaN	0.13102504	NaN

Table 2.14: Means of the metrics after 50 simulations ($n < p$, low sparsity) for the AIC, one standard error rule tuning λ for the node wise lasso and the graphical lasso

Method	Sensitivity	FPR	Precision	G-mean	F_1-Score
AIC (SE RULE) (E1) - SE	0.0005742638	0.0030388601	NaN	0.0021192393	NaN
AIC (SE RULE) (E2) - SE	0.0007521446	0.0039760513	0.0030569730	0.0015980578	0.0012061748
AIC (SE RULE) (E3) - SE	0.0052242975	0.0044850584	NaN	0.0043016177	NaN

Table 2.15: Standard error of the means of the metrics after 50 simulations ($n < p$, low sparsity) for the AIC, one standard error rule tuning λ for the node wise lasso and the graphical lasso

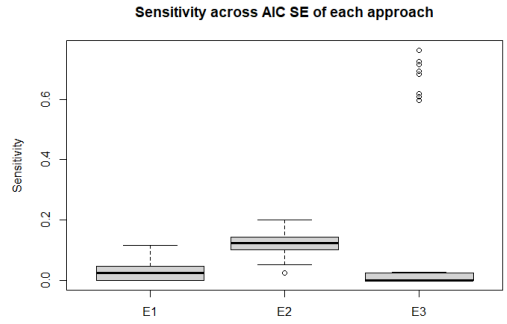

Figure 2.36: Box-plot of the Sensitivity for each of the approaches for Case 6 ($n < p$, low sparsity)

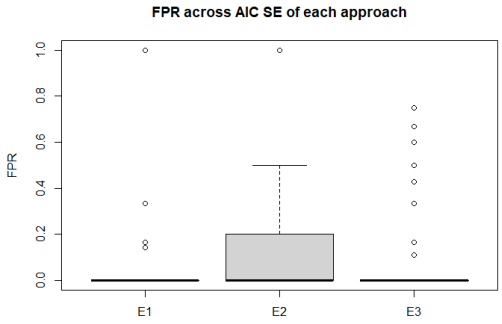

Figure 2.37: Box-plot of the FPR for each of the approaches for Case 6 ($n < p$, low sparsity)

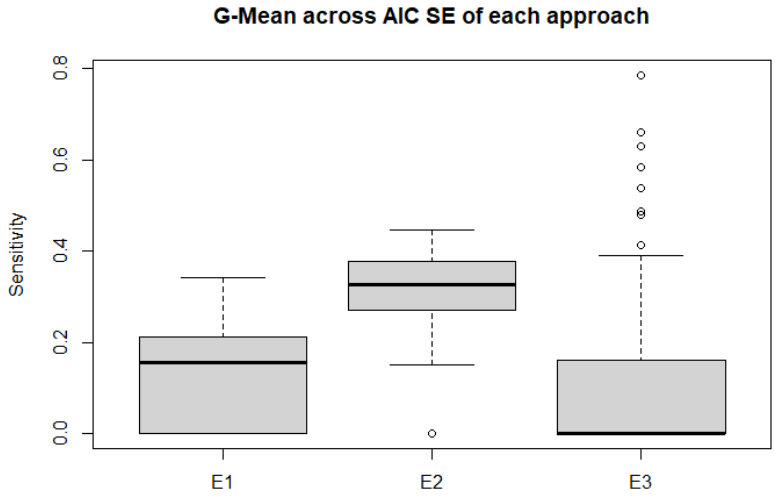

Figure 2.38: Box-plot of the $G - Mean$ for each of the approaches for Case 6 ($n < p$, low sparsity)

2.7 Conclusion

The node wise lasso and the graphical lasso approaches are considered useful when estimating the true set of connected edges E, which summarises the dependence relationship between the number of variables. From the results obtained after performing the simulations, the best method for both node wise lasso approaches in recovering the true set E, was when tuning λ by the AIC one standard error rule approach; while for the graphical lasso the 5-CV one standard error rule was preferred. However, the graphical lasso tend to give a worst estimate than the two node wise lasso approaches. Nevertheless, this simulations were performed in the case when $n > p$ and where the true network had a high sparsity. As a further research, more simulations settings can be performed.

After performing the simulations for each set, we decided to focus on one of the methods for tuning the parameter λ and apply it for the node wise lasso and graphical lasso approaches in different simulation settings. This method was chosen to be the AIC one standard error

rule. For the case $n > p$ with medium or low sparsity, the methods tend to perform better and obtain a higher accuracy in recovering the true set E when the setting of high sparsity is considered. However, for the case $n < p$ the results obtained tend to be very similar for the different sparsity levels and tend to give very poor performance. This can be due to the excessive shrinkage applied for the high dimensionality and due to the consideration of the one standard error rule λ as our tuning parameter. Improvements can be done in this setting by considering other approaches in the tuning of λ, which examine smaller values of this parameter to reduce the shrinkage.

Further research can be done by applying more tuning parameter λ methods in different simulations settings to each of the node wise lasso and graphical lasso approaches in order to obtain accurate estimates of the true set E. Another curious thing that seemed to keep appearing when applying the different simulation settings using the AIC one standard error rule to estimate λ, was the ocurrence of outliers in the Sensitivity for the graphical lasso. These outlying Sensitivities obtained values very close to one even for the $n < p$ cases, and it would be interesting to further analyse these observations and their capacity to obtain a large Sensitivity value even when excessive shrinkage is applied.

Bibliography

Angelini, C., D. De Canditiis, and A. Plaksienko (2021). Jewel: A novel method for joint estimation of gaussian graphical models. *Mathematics 9*(17), 2105.

Banerjee, O., L. El Ghaoui, and A. d'Aspremont (2008). Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data. *The Journal of Machine Learning Research 9*, 485–516.

Fan, J., Y. Feng, and L. Xia (2020). A projection-based conditional dependence measure with applications to high-dimensional undirected graphical models. *Journal of Econometrics 218*(1), 119–139.

Friedman, J., T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso.

Friedman, J., T. Hastie, and R. Tibshirani (2010). Applications of the lasso and grouped lasso to the estimation of sparse graphical models. Technical report, Technical report, Stanford University.

Hoo, Z. H., J. Candlish, and D. Teare (2017). What is an roc curve?

Hossin, M. and M. Sulaiman (2015). A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process 5*(2), 1.

James, G., D. Witten, T. Hastie, and R. Tibshirani (2013). *An introduction to statistical learning*, Volume 112. Springer.

Luque, A., A. Carrasco, A. Martín, and A. de las Heras (2019). The impact of class imbalance in classification performance metrics based on the binary confusion matrix. *Pattern Recognition 91*, 216–231.

Marzban, C. (2004). The roc curve and the area under it as performance measures. *Weather and Forecasting 19*(6), 1106–1114.

Mazumder, R. and T. Hastie (2012). The graphical lasso: New insights and alternatives. *Electronic journal of statistics 6*, 2125.

Meinshausen, N. and P. Bühlmann (2006). High-dimensional graphs and variable selection with the lasso. *The annals of statistics 34*(3), 1436–1462.

Tharwat, A. (2020). Classification assessment methods. *Applied Computing and Informatics*.

Yuan, M. and Y. Lin (2007). Model selection and estimation in the gaussian graphical model. *Biometrika 94*(1), 19–35.

Appendix A

Real World Data

A.1 Dataset Description

popularity_data

	track_popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness
1	66	0.642039314610822	1.20159575777518	0.17319726714015	1.3671025927324	0.876163586990853	-0.48135504873698	-0.333892760051908	-0.377947261566378
2	67	0.490404401222476	0.643307210635269	1.55760321514204	0.585757535270544	0.876163586990853	-0.688631167939184	-0.468663317998463	-0.359171907369673
3	70	0.138887102004039	1.28450989843952	-1.21120868086174	1.10007331827649	-1.1413046121585	-0.32441741562674	-0.436791902267858	-0.377843350461204
4	60	0.435264432717623	1.2789822890619	0.450078456740527	0.984293708048734	0.876163586990853	-0.0500233149685853	-0.667632013345235	-0.377905206556859
5	69	-0.0334252995736271	0.74280417943248	-1.21120868086174	0.685140611101881	0.876163586990853	-0.702449575885997	-0.432694148816781	-0.377947261566378
6	67	0.138887102004039	1.21817858590805	0.726959646340905	0.446554304476483	0.876163586990853	0.196733969795943	-0.434515372572815	-0.377947261566378
7	62	-1.41881700825806	0.869939195117806	-0.103683922460227	0.646324325591999	-1.1413046121585	-0.441873883174656	0.0531172881054293	-0.377947261566378
8	69	-0.777814874389142	1.12973683586608	-0.380565112060605	1.43904657018606	-1.1413046121585	-0.628422390456639	-0.64577732827282	-0.377925721195649
9	68	-0.419405079107598	1.30662033595001	0.726959646340905	1.05657230865334	0.876163586990853	-0.499121573240026	-0.684933639027563	-0.377929556541162
10	67	-0.0885652680784801	0.659890038768138	-0.934327491261359	0.725295389215553	0.876163586990853	-0.740943712309263	-0.540146350422818	-0.377947261566378
11	58	0.166457086256465	1.24028902341854	0.17319726714015	0.0734494911702858	0.876163586990853	0.729729704887323	-0.13355814688811	-0.377925319822281
12	67	-1.50152696101534	0.416675226152732	0.726959646340905	0.602823315968855	0.876163586990853	-0.509978893769665	-0.124452028107937	-0.377947261566378
13	67	0.614469330358395	0.151349976026835	-1.21120868086174	0.684136741649039	0.876163586990853	-0.599798545423953	-0.616637748176268	-0.377947261566378
14	68	-0.571039992495944	1.19606814839755	-0.103683922460227	0.759092327461226	-1.1413046121585	-0.439899824896539	-0.747765858610754	-0.377947261566378
15	63	0.242274542950637	0.44984088241847	-0.103683922460227	0.760765443215963	-1.1413046121585	-0.470497728207341	-0.764930892511379	-0.369786003091326
16	66	1.0349115902079	0.753859398187726	-1.48808987046211	0.708229608517242	0.876163586990853	-0.172414928211791	-0.206407097129491	-0.377924829254832
17	60	0.269844527203064	1.11868161711083	-1.21120868086174	0.802258713933423	-1.1413046121585	-0.121089412980769	-0.478680048656653	-0.377947261566378
18	65	0.159564590193359	0.267429772956915	1.28072202554166	0.478678126967421	-1.1413046121585	0.571805042638025	-0.618458971932302	-0.377947261566378
19	69	0.628254322484609	-0.782816008791426	-1.21120868086174	-0.000836848340006541	0.876163586990853	-0.521823243438363	-0.329339700661821	-0.361892326861357
20	35	-1.29475207912214	0.676472866901006	0.450078456740527	0.419449829249755	0.876163586990853	-0.134907820927583	-0.762290118065129	-0.375980532065013
21	70	-0.571039992495944	1.30109272657239	0.17319726714015	0.557649190590974	-1.1413046121585	-0.616578040787941	-0.78500988442166	-0.289645120688763
22	64	0.366339472086557	1.18501292964231	1.00384083594128	0.799916351876792	-1.1413046121585	-0.713306896415636	-0.39308253212303	-0.0167112307034056
23	62	0.200919566571998	0.659890038768138	-0.380565112060605	0.750392125536597	-1.1413046121585	0.206604261186524	-0.584766332445664	-0.377932143169531
24	70	0.531759377601116	0.836773538852069	-1.21120868086174	0.241095689794864	0.876163586990853	-0.618552099066057	-0.279256047370872	-0.377938252964127
25	58	-0.240200181466826	1.05235030457936	1.55760321514204	0.981951345992103	0.876163586990853	-0.709358779859404	-0.724089949782305	-0.377864311070402
26	63	0.138887102004039	0.339288694866012	1.55760321514204	-0.106243140888394	0.876163586990853	-0.628422390456639	-0.324786641271735	-0.377947261566378

	liveness	valence	tempo	duration_ms	year	playlist_genreedm	playlist_genrelatin	playlist_genrepop	playlist_genrer&b	playlist_genrerap	playlist_genrerock
1	-0.809217191381052	0.0319071616168705	0.042926129610997	-0.518865668600007	0.688658903028329	0	0	1	0	0	0
2	1.08104423254507	0.782509870813304	-0.777186427036735	-1.05625238314474	0.688658903028329	0	0	1	0	0	0
3	-0.51955423063543	0.439377203752077	0.116224804343852	-0.822004320973346	0.688658903028329	0	0	1	0	0	0
4	0.0895804071741546	-1.00177999790508	0.0399525524386702	-0.947735497608291	0.688658903028329	0	0	1	0	0	0
5	-0.692574388396238	0.919762937637795	0.115035373514922	-0.614162647088969	0.688658903028329	0	0	1	0	0	0
6	-0.305709091829938	0.319280770280648	0.152428105199431	-1.04874828925811	0.688658903028329	0	0	1	0	0	0
7	-0.0918639530244451	-1.53792479018824	-0.306023139926553	-0.637176315868221	0.688658903028329	0	0	1	0	0	0
8	-0.513074074914052	-0.615755747461195	0.262227438595099	-0.303854158908586	0.688658903028329	0	0	1	0	0	0
9	2.89548783453107	-0.620044905799461	0.227994132549936	-0.545054789135294	0.688658903028329	0	0	1	0	0	0
10	-0.636845049192382	0.340726561971975	0.151498862364329	0.455262652727174	0.688658903028329	0	0	1	0	0	0
11	-0.42883205053613	1.03557021277096	0.0409933044139843	-0.299258110314327	0.688658903028329	0	0	1	0	0	0
12	-0.370510649043723	-0.778743764315278	0.0834039474080455	-0.368800503771459	0.688658903028329	0	0	1	0	0	0
13	1.19120687980851	0.756774920783713	0.0410304741273885	-0.556419563841098	0.688658903028329	0	0	1	0	0	0
14	0.964401429560261	0.718172495739325	0.112916699850889	-1.02423045912074	0.688658903028329	0	0	1	0	0	0
15	-0.759968007898575	-1.16905717309742	0.192868753383076	0.491997615673287	0.688658903028329	0	0	1	0	0	0
16	1.1328854783161	0.906895462622999	0.154137912016019	-0.627900654159808	0.688658903028329	0	0	1	0	0	0
17	1.53465513304157	-0.61146658912293	-0.105195178404282	-0.873279510381296	0.688658903028329	0	0	1	0	0	0
18	-0.104824264467202	0.0233288449403399	-0.0326770675529122	-0.608764379103712	0.601073682549593	0	0	1	0	0	0
19	-0.33810987043683	-0.800189556006605	-0.33155873303516	-1.10966682426202	0.688658903028329	0	0	1	0	0	0
20	-0.38347096048648	-1.19479212312702	0.0667147460896114	-0.675766411156017	0.688658903028329	0	0	1	0	0	0
21	-0.551955009242323	-1.31488855659845	0.188334048347777	-0.176518543491646	0.688658903028329	0	0	1	0	0	0
22	-0.500113763471295	0.842558087549019	0.52538900949602	-0.921429386745151	0.688658903028329	0	0	1	0	0	0
23	-0.461232829143023	-0.748719655947421	0.113622924405567	-0.247882643482502	0.688658903028329	0	0	1	0	0	0
24	-0.980941317997584	-0.594309955769869	0.116819519758318	-0.53902143079883	0.688658903028329	0	0	1	0	0	0
25	1.32729014995746	-0.379852038856602	-0.403779486179297	-0.346070954359852	0.688658903028329	0	0	1	0	0	0
26	0.906080028067854	-0.182550755296396	0.150606789242631	-0.885780762557681	0.688658903028329	0	0	1	0	0	0

Figure A.1: Overview of the Popularity Dataset

playlist_genre	danceability	energy	loudness	speechiness	instrumentalness	liveness	valence	tempo
pop	0.0946248130643341	0.217627486068002	4.18377666478555	-0.0496392423814898	-0.0911294458750705	0.0653	0.00761448045993229	1.07778079147856
pop	0.0726248130643341	0.116627486068002	1.84877666478555	-0.0706392423814898	-0.0869194458750705	0.357	0.182614480459932	-20.9862192085214
pop	0.0216248130643342	0.232627486068002	3.38577666478555	-0.0337392423814898	-0.0911061458750705	0.11	0.102614480459932	3.04978079147855
pop	0.0646248130643341	0.231627486068002	3.03977666478555	-0.00593924238148985	-0.0911200158750705	0.204	-0.233385519540068	0.997780791478561
pop	-0.00337518693566585	0.134627486068002	2.14577666478555	-0.0720392423814898	-0.0911294458750705	0.0833	0.214614480459932	3.01778079147856
pop	0.0216248130643342	0.220627486068002	1.43277666478555	0.0190607576185102	-0.0911294458750705	0.143	0.0746144804599322	4.02378079147856
pop	-0.204375186935666	0.157627486068002	2.02977666478555	-0.0456392423814898	-0.0911294458750705	0.176	-0.358385519540068	-8.31021920852145
pop	-0.111375186935666	0.204627486068002	4.39877666478555	-0.0645392423814898	-0.0911246158750705	0.111	-0.143385519540068	6.97778079147857
pop	-0.0593751869356659	0.236627486068002	3.25577666478555	-0.0514392423814898	-0.0911254758750705	0.637	-0.144385519540068	6.05678079147856
pop	-0.0113751869356659	0.119627486068002	2.26577666478555	-0.0759392423814898	-0.0911294458750705	0.0919	0.0796144804599322	3.99878079147855
pop	0.0256248130643342	0.224627486068002	0.317776664785553	0.0730607576185101	-0.0911245258750705	0.124	0.241614480459932	1.02578079147855
pop	-0.216375186935666	0.0756274860680023	1.89977666478555	-0.0525392423814898	-0.0911294458750705	0.133	-0.181385519540068	2.16678079147856
pop	0.0906248130643341	0.0276274860680022	2.14277666478555	-0.0616392423814898	-0.0911294458750705	0.374	0.176614480459932	1.02678079147856
pop	-0.0813751869356659	0.216627486068002	2.36677666478555	-0.0454392423814898	-0.0911294458750705	0.339	0.167614480459932	2.96078079147856
pop	0.0366248130643341	0.0816274860680023	2.37177666478555	-0.0485392423814898	-0.0892994458750705	0.0729	-0.272385519540068	5.11178079147855
pop	0.151624813064334	0.136627486068002	2.21477666478555	-0.0183392423814898	-0.0911244158750705	0.365	0.211614480459932	4.06978079147856

Figure A.2: Overview of the Genre Dataset

A.2 Data Preprocess Code

The code is in R.

COLUMN NAME	COLUMN DESCRIPTION
TRACK_POPULARITY	Song popularity (0~100) where higher is better
DANCEABILITY	Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable
ENERGY	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity.
KEY	The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation.
LOUDNESS	The overall loudness of a track in decibels (dB). Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.
MODE	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
SPEECHINESS	Speechiness detects the presence of spoken words in a track.
ACOUSTICNESS	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
INSTRUMENTALNESS	The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content.
LIVENESS	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live.
VALENCE	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track.
TEMPO	In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
DURATION_MS	Duration of song in milliseconds
YEAR	Year when album released
PLAYLIST_GENRE	Playlist genre

Figure A.3: Column Description of Popularity Dataset

```
spotify_songs <- readr::read_csv('https://raw.githubusercontent.com/
rfordatascience/tidytuesday/master/data/2020/2020-01-21/spotify_songs.csv')

colnames(spotify_songs)
dim(spotify_songs)
# remove the duplicate
data=spotify_songs[!duplicated(spotify_songs),]
dim(data)
# weather it has NA
sum(is.na(data_1))
#drop some lists
droplist=c("track_id","track_name","track_artist","track_album_id",
"track_album_name","playlist_subgenre","playlist_name","playlist_id")

data_1=data[,!(colnames(data) %in% droplist)]
dim(data_1)
```

COLUMN NAME	COLUMN DESCRIPTION
POPULIST_GENRE	The genre of the songs
DANCEABILITY	It describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
ENERGY	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity.
LOUDNESS	Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.
SPEECHINESS	Speechiness detects the presence of spoken words in a track.
INSTRUMENTALNESS	The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content.
LIVENESS	Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
VALENCE	Tracks with high valence sound more positive, while tracks with low valence sound more negative
TEMPO	In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.

Figure A.4: Column Description of Genre Dataset

```
colnames(data_1)
#process the date: only use the year
library(stringr)
y=c()
date=data_1['track_album_release_date']
for(i in 1:dim(date)[1]){
y[i]=str_split_fixed(date[i,1],'-',n=3)[1]
}

year=as.integer(y)
typeof(year)
length(year)
dim(data_1)
data_2=data_1[-2]
df=cbind(data_2,year)
dim(df)
```

```
#get the label of the data
X=df[-1]
Y=df[1]
colnames(X)
#View(X)
summary(X)
# standardize and centralize
std=scale(X[-1],center=TRUE,scale=TRUE)
summary(std)
colnames(std)
dim(std)
#do one-hot for the genre
genre=model.matrix(~playlist_genre-1,df) %>% as.data.frame()
dim(genre)
data_clean=cbind(std,as.data.frame(model.matrix(~playlist_genre-1,df)))
dim(data_clean)
colnames(data_clean)
head(data_clean)

#create dataset for regression
data_process=cbind(Y,data_clean)
summary(data_process)
dim(data_process)
colnames(data_process)
View(data_process)
# already process the data: data_process,
split it into train and test data: train and test
index=sort(sample(nrow(data_process),nrow(data_process)*0.7))
train=data_process[index,]
test=data_process[-index,]

#create new column popular for binary classify
popular_value=data_process['track_popularity']
popular = rep(0, 32833)
popular[popular_value > 50] = 1
summary(popular)
data_binary=cbind(popular,data_clean)
# generate train and test dataset
train_binary=data_binary[index,]
test_binary=data_binary[-index,]
train_binary_X=train_binary[-1]
train_binary_Y=train_binary[1]
test_binary_X=test_binary[-1]
test_binary_Y=test_binary[1]
colnames(train_binary)
dim(train_binary)
dim(test_binary)
```

A.3 Binary Classification

Classification	Accuracy	True Positive rate	False Positive rate
Random Forest	0.7102538	0.6990569	0.2835774
Bagging	0.7077157	0.6858696	0.2792545
Boost	0.6859898	0.6416298	0.2827216
KNN ($k = 37$)	0.6392893	0.5925229	0.3323549
LDA	0.6172589	0.419005	0.2438401
Logistic Regression	0.6153299	0.4244282	0.240328
Classification Tree	0.588731	0.5277149	0.3770985

Table A.1: Binary Classification summary

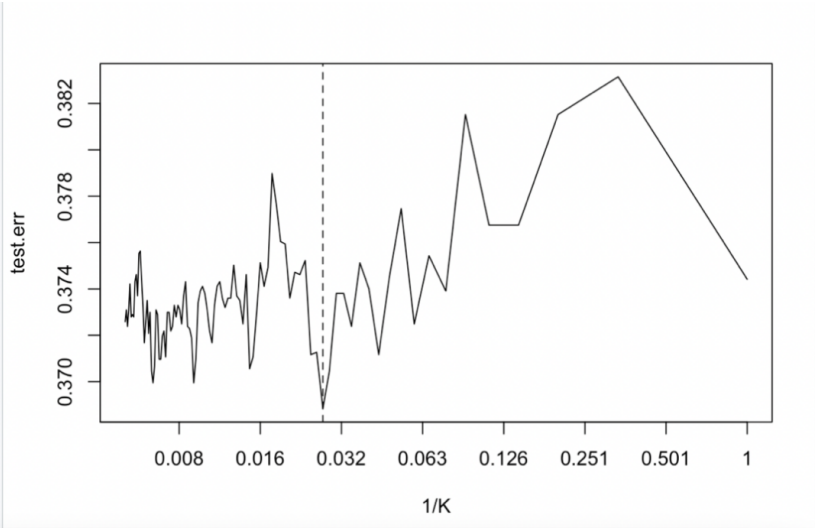

Figure A.5: Misclassification error rate against $1/K$

```
#binary classify

#logistic regression
glm.bin=glm(popular~.,data=train_binary,family = binomial)
summary(glm.bin)
glm_bin_probs = predict(glm.bin,test_binary,type='response')

glm_bin_pred = rep(0,9850)
length(glm_bin_probs)
glm_bin_pred[glm_bin_probs > 0.5] = 1
glm_cm=table( test_binary$popular,glm_bin_pred)
mean(glm_bin_pred == test_binary$popular)
tpr=1800/(1800+2441)

fpr=glm_cm[1,2]/sum(glm_cm[1,])
library(ROCR)
pred=prediction(glm_bin_probs,test_binary$popular)
auc=performance(pred,'auc')@y.values
perf=performance(pred,'tpr','fpr')
plot(perf, main = "ROC curve",sub="Logistic Regression")
```

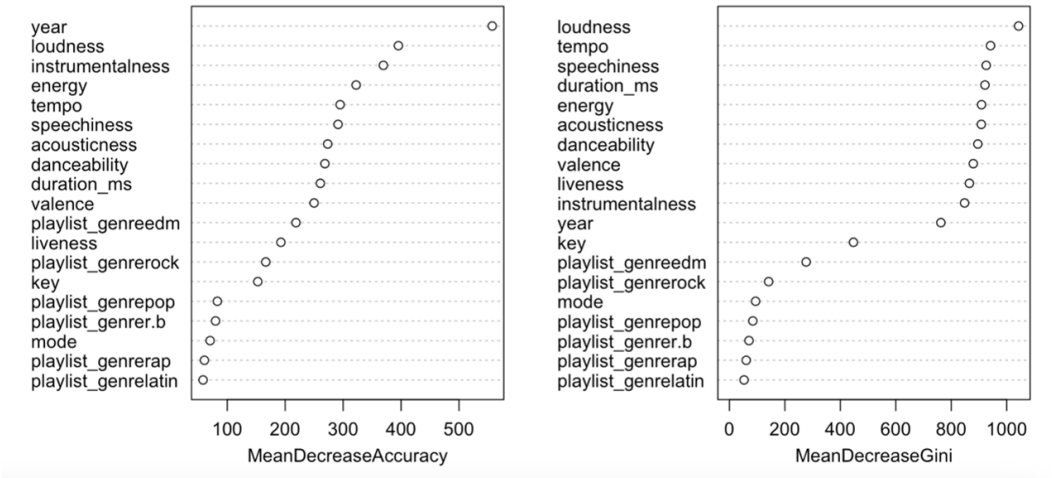

Figure A.6: Variable importance by bagging

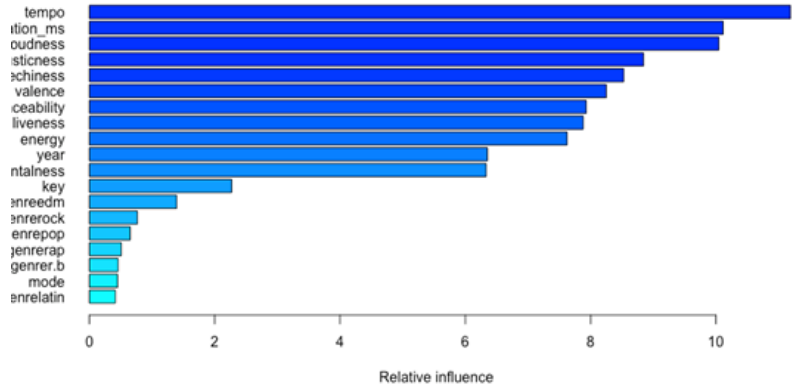

Figure A.7: Variable importance by Boost

```
library(pROC)
modelroc=roc(test_binary$popular,glm_bin_probs)

plot(modelroc,print.auc=TRUE,auc.polygon=TRUE,max.auc.polygon=TRUE,
     auc.polygon.col='skyblue',orint.thres=TRUE,main="Logistic_Regression")



#Linear Discriminant Analysis
library(MASS)
lda.bin=lda(popular~.,data=train_binary)
lda.bin
lda_bin_pre=predict(lda.bin,test_binary)
summary(lda_bin_pre)
lda_cm=table(test_binary$popular,lda_bin_pre$class)
mean(lda_bin_pre$class==test_binary$popular)#accuacy
```

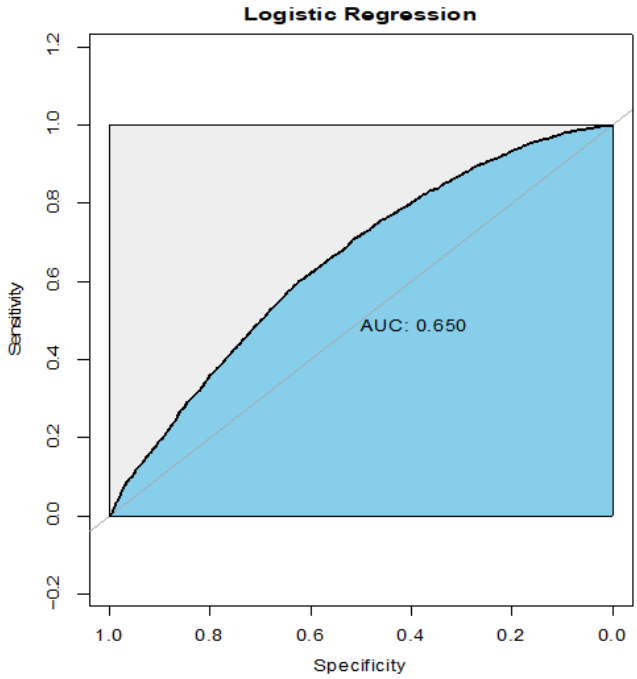

Figure A.8: *ROC* curve for Logistic Regression

```
fpr=lda_cm[1,2]/sum(lda_cm[1,])
tpr=lda_cm[2,2]/sum(lda_cm[2,])

pred=prediction(lda_bin_pre$x,test_binary$popular)
auc=performance(pred,'auc')@y.values
perf=performance(pred,'tpr','fpr')
par(new=TRUE)
plot(perf, main = "ROC curve",sub=auc, colorize = T)

modelroc=roc(test_binary$popular,lda_bin_pre$x)

plot(modelroc,print.auc=TRUE,auc.polygon=TRUE,max.auc.polygon=TRUE,
     auc.polygon.col='skyblue',orint.thres=TRUE,main="LDA")


#KNN
library(class)
knn_pred = knn(train_binary_X, test_binary_X, train_binary$popular, k = 1)
length(knn_pred)
knn_cm=table(knn_pred, test_binary$popular)
mean(knn_pred == test_binary$popular)#accuracy

knn_pred = knn(train_binary_X, test_binary_X, train_binary$popular, k = 3)
length(knn_pred)
knn_cm=table(knn_pred, test_binary$popular)
mean(knn_pred == test_binary$popular)#accuracy

#get the error for different K
```

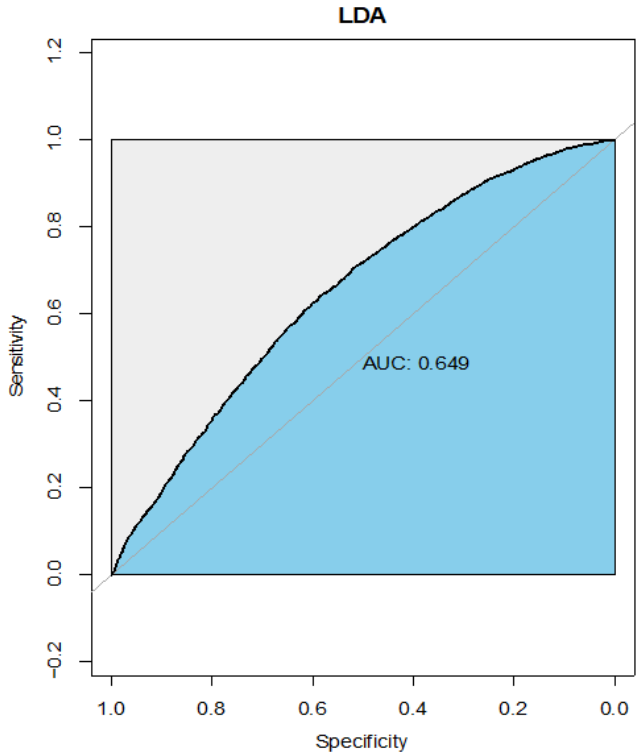

Figure A.9: *ROC* curve for *LDA*

```
K=seq(1,200,by=2)
nk=length(K)
knn.pred=as.data.frame(matrix(NA,nrow(test_binary),nk))
test.err=rep(NA,nk)
for(i in 1:nk){
  knn.pred[,i]=knn(train_binary_X, test_binary_X, train_binary$popular,
  k = K[i])
  test.err[i]=mean(knn.pred[,i]!=test_binary_Y$popular)
}
#plot error-1/k:
plot(x=1/K,y=test.err,type='l',log='x',xaxt='n')
ticks=exp(seq(log(0.002),log(1),length.out=10))
axis(1,at=ticks,labels=round(ticks,3))
k_min=K[which.min(test.err)]
abline(v=1/k_min,lty=2)

knn_pred=knn(train_binary_X, test_binary_X, train_binary$popular,
             k = k_min,prob=T)
mean(knn_pred==test_binary$popular)
knn_cm=table(knn_pred, test_binary$popular)
fpr=knn_cm[1,2]/sum(knn_cm[1,])
tpr=knn_cm[2,2]/sum(knn_cm[2,])
knn_prob = attributes(knn_pred)$prob

pred=prediction(knn_prob,test_binary$popular)
auc=performance(pred,'auc')@y.values
```

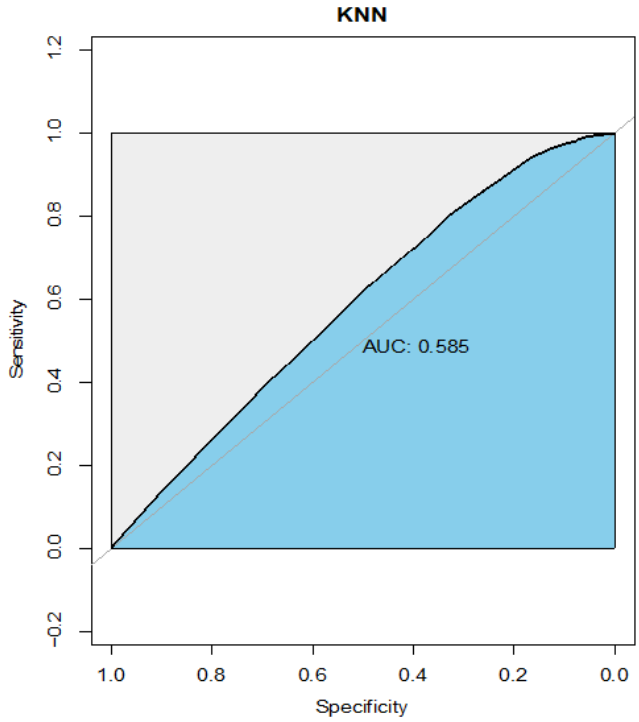

Figure A.10: *ROC* curve for KNN ($K = 37$)

```
perf=performance(pred,'tpr','fpr')
plot(perf, main = "ROC curve", colorize = T)

modelroc=roc(test_binary$popular,knn_prob)

plot(modelroc,print.auc=TRUE,auc.polygon=TRUE,max.auc.polygon=TRUE,
     auc.polygon.col='skyblue',orint.thres=TRUE,main="KNN")



#classification tree
library(tree)
set.seed(1)

train_binary$popular=as.factor(train_binary$popular)
popular_test=as.factor(test_binary$popular)
train_binary=data.frame(train_binary)
tree_bin= tree(popular ~., data = train_binary)
plot(tree_bin)
text(tree_bin,pretty=0,cex=0.8)
summary(tree_bin)
test_binary=data.frame(test_binary)
tree_pred_binary=predict(tree_bin,test_binary,type='class')
tree_cm=table(tree_pred_binary,test_binary$popular)
mean(tree_pred_binary==test_binary$popular)

fpr=tree_cm[1,2]/sum(tree_cm[1,])
```

```
tpr=tree_cm[2,2]/sum(tree_cm[2,])

cv_tree_bin=cv.tree(tree_bin,FUN=prune.misclass)
names(cv_tree_bin)
par(mfrow=c(1,2))
plot(cv_tree_bin$size,cv_tree_bin$dev,type="b")
plot(cv_tree_bin$k, cv_tree_bin$dev, type="b")


#bagging
library(randomForest)
set.seed(1)
dim(data_binary)
bog_bin=randomForest(popular~.,data=train_binary,mtry=19,importance=TRUE,
                     ntree=5000)

bag_pred=predict(bog_bin,newdata = test_binary)

importance(bog_bin)
varImpPlot(bog_bin)

mean(bag_pred==test_binary$popular)
bag_cm=table(bag_pred,test_binary$popular)
fpr=bag_cm[1,2]/sum(bag_cm[1,])
tpr=bag_cm[2,2]/sum(bag_cm[2,])

#randomforest
rf_bin=randomForest(popular~.,data=train_binary,mtry=5,importance=TRUE,
                    ntree=5000)

rf_pred=predict(rf_bin,newdata = test_binary)
rf_cm=table(rf_pred,test_binary$popular)
mean(rf_pred==test_binary$popular)
fpr=rf_cm[1,2]/sum(rf_cm[1,])
tpr=rf_cm[2,2]/sum(rf_cm[2,])
importance(rf_bin)
varImpPlot(rf_bin)


#boost
library(gbm)
set.seed (1)
boost.bin = gbm( popular ~ ., data = train_binary,
                 distribution = "multinomial",
                   n.trees = 5000, interaction.depth = 4)
summary(boost.bin,las=1)

boost_pred=predict(boost.bin,newdata = test_binary,n.trees = 5000)
boost_pred=colnames(boost_pred)[apply(boost_pred,1,which.max)]
```

```
mean(boost_pred==test_binary$popular)
boost_cm=table(boost_pred,test_binary$popular)

fpr=boost_cm[1,2]/sum(boost_cm[1,])
tpr=boost_cm[2,2]/sum(boost_cm[2,])
```

A.4 Multi-Class Classification

Classification	Prediction Accuracy
Logistic Regression	0.45434606
LDA	0.44987600
$kNN(k = 15)$	0.3773606
QDA $(k = 37)$	0.4561456
Boosting	0.7107395
Random Forest	0.9066431
Bagging	0.8985442
SVM	0.6065632
Tree	0.5571002
Prune Tree $(k = 7)$	0.5571002

Table A.2: Muti-class Classification accuracy summary

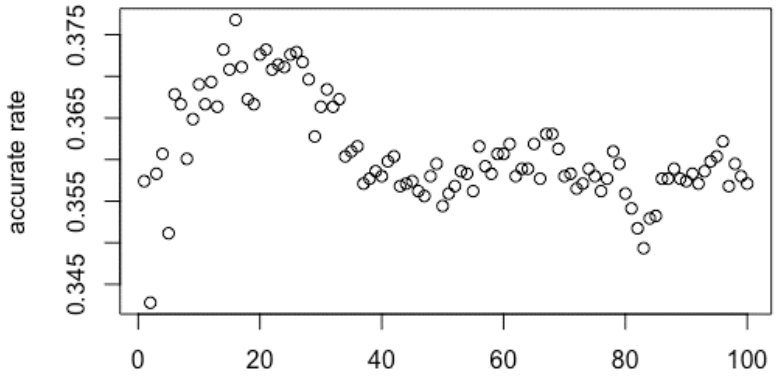

Figure A.11: Knn plot: optimal at $k = 15$

```
#data cleaning#
y<-read.csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday
/master/data/2020/2020-01-21/spotify_songs.csv')

index<-duplicated(y$track_id)
y1<-y[!index,]
dim(y1)
y2=na.omit(y1)
dim(y2)
names(y2)
y3<-data.frame(y2[10],y2[12:13],y2[15],y2[17],y2[19:22])
y3$playlist_genre<-factor(y3$playlist_genre)
```

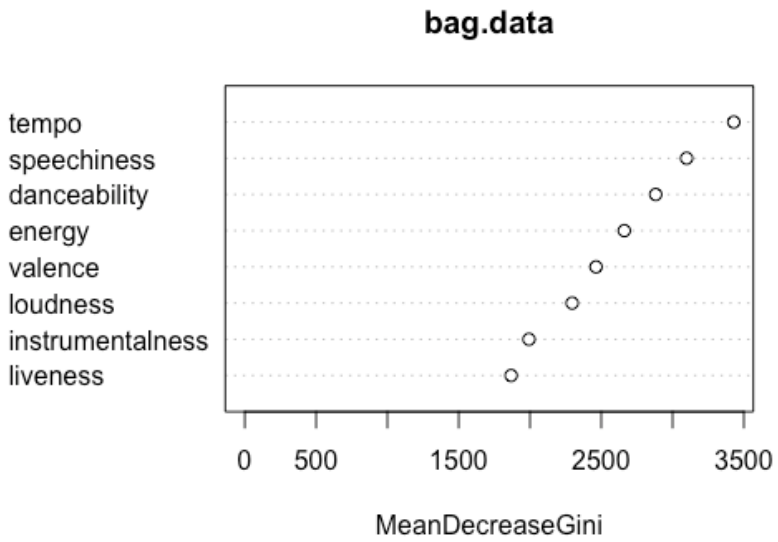

Figure A.12: Bagging importance plot

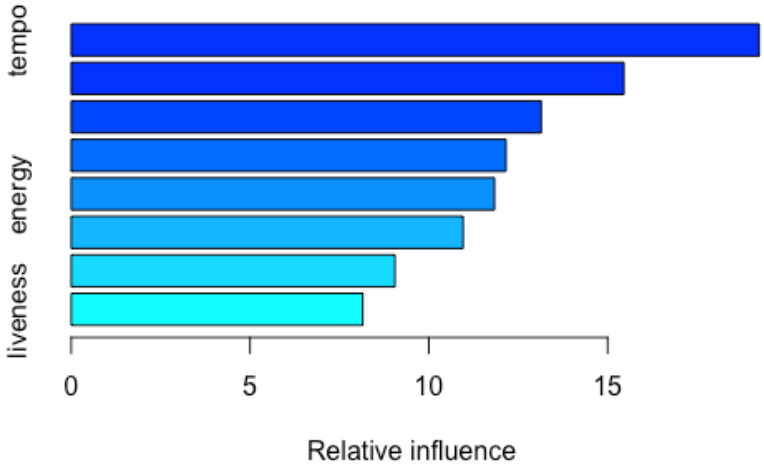

Figure A.13: Boosting importance summary

```
names(y3)
#standardization#
data_stand<-function(c){
return(c-mean(c))/(sd(c)/sqrt(length(c)))
}
y3$danceability=data_stand(y3$danceability)
y3$energy=data_stand(y3$energy)
y3$loudness=data_stand(y3$loudness)
y3$speechiness=data_stand(y3$speechiness)
y3$instrumentalness=data_stand(y3$instrumentalness)
y3$valence=data_stand(y3$valence)
y3$tempo=data_stand(y3$tempo)

# method 1 multinomial logistic regression#
attach(y3)
library(nnet)
```

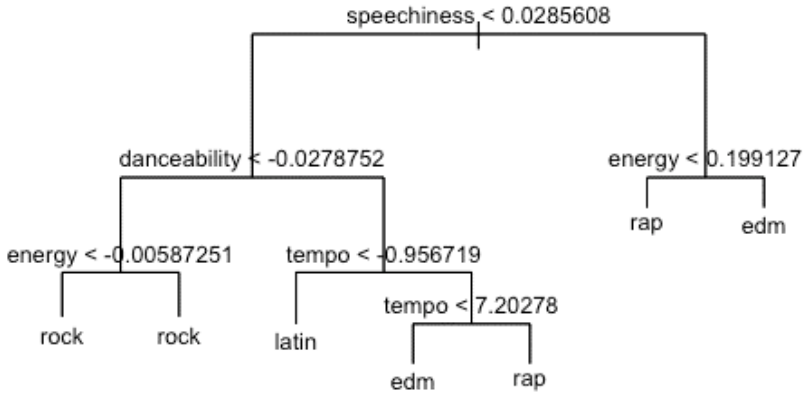

Figure A.14: Tree

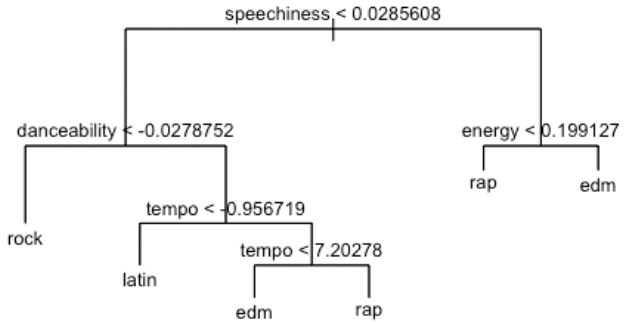

Figure A.15: Prune Tree

```
library(caret)
train_sub=sample(nrow(y3),25000)
train_data=y3[train_sub,]
test_data=y3[-train_sub,]
mult.model<-multinom(playlist_genre~.,data=train_data)
pre_logistic<-predict(mult.model,newdata=test_data,type='class')
#confusin matrix and accuracy#
table(pre_logistic,test_data$playlist_genre)
mean(pre_logistic==test_data$playlist_genre)

#method2 knn#
library(class)
#write a function where the both confusion #matrix and prediction accuracy
#are displayed
knn_test<-function(v){

    knn_train<-cbind(train_data$danceability,train_data$energy,
    train_data$loudness,train_data$speechiness,train_data$instrumentalness,
    train_data$liveness,train_data$valence,train_data$tempo)

    knn_test<-cbind(test_data$danceability,test_data$energy,
    test_data$loudness, test_data$speechiness,test_data$instrumentalness,
    test_data$liveness,test_data$valence,test_data$tempo)
```

```
    knn_pred<-knn(knn_train,knn_test,train_data$playlist_genre,k=v)
    table(knn_pred,test_data$playlist_genre)
    mean(knn_pred==test_data$playlist_genre)
}
knn.final<-c()
#iterate through different k to see how accuracy changes
for(i in 1:100){
    knn.final[i]<-knn_test(i)
}
knn.final
#plot the relationship between k and accuracy rate

plot(knn.final,xlab='k',ylab='accurate_rate')
#find the k with max accuracy, k=15indicates data very flexible
max(knn.final)
which(knn.final==max(knn.final))

#method 3 lda&qda#
library(MASS)
lda_fit<-lda(playlist_genre~.,data=train_data)
lda_predict<-predict(lda_fit,test_data)
table(lda_predict$class,test_data$playlist_genre)
mean(lda_predict$class==test_data$playlist_genre)

qda_fit<-qda(playlist_genre~.,data=train_data)
qda_predict<-predict(qda_fit,test_data)
table(qda_predict$class,test_data$playlist_genre)
mean(qda_predict$class==test_data$playlist_genre)

#method 4 tree and prune tree#
library(tree)
tree.train<-tree(playlist_genre~.,data=train_data)
tree.pred<-predict(tree.train,test_data,type="class")
table(tree.pred==test_data$playlist_genre)
mean(tree.pred==test_data$playlist_genre)
plot(tree.train)
text(tree.train,pretty=0,cex=0.8)
set.seed(7)
#prune tree#
cv.data<-cv.tree(tree.train,FUN=prune.misclass)
cv.data
#best size is 7 same as tree#
prune.data<-prune.misclass(tree.train,best=6)
plot(prune.data)
text(prune.data,pretty=0,cex=0.8)
tree.prunepred<-predict(prune.data,test_data,type="class")
table(tree.prunepred,test_data$playlist_genre)
mean(tree.prunepred==test_data$playlist_genre)
```

```
#method 5 boosting#
set.seed(7)
library(gbm)
boost=gbm(playlist_genre~.,data=train_data,distribution="multinomial",
          n.trees=5000,interaction.depth=4)

boost.predict<-predict(boost,test_data,n.trees=5000,type='response')
boost.predict
boost.predict_class=colnames(boost.predict)[apply(boost.predict,1,which.max)]

mean(boost.predict_class==test_data$playlist_genre)
summary(boost)#tempo is most important#

#method 6 random forest#
library(randomForest)
rf.fit<-randomForest(playlist_genre~.,data=train_data,mtry=3)
rf.predict<-predict(rf.fit,test_data,type='class')
table(rf.predict,test_data$playlist_genre)
mean(rf.predict==test_data$playlist_genre)

#method 7 bagging#
bag.data<-randomForest(playlist_genre~.,data=train_data,mtry=7)
bag.predict<-predict(bag.data,newdata=test_data)
mean(bag.predict==test_data$playlist_genre)
importance(bag.data)
varImpPlot(bag.data)#tempo is still the most important#

#method 8 svm#
library(e1071)
train_data$playlist_genre<-factor(train_data$playlist_genre)
model<-svm(playlist_genre~.,data=train_data)
test_data$playlist_genre<-factor(test_data$playlist_genre)
pred<-predict(model,test_data)
table(pred,test_data$playlist_genre)
mean(pred==test_data$playlist_genre)

# We also tried calculating the AUC
#but accuracy rate is more useful in answering
#the questions we are interested in
#example of code below
library(pROC)
qda_prob<-predict(qda_fit,test_data,type='prob')
qda.roc<-multiclass.roc(test_data$playlist_genre,qda_prob$posterior)
qda.roc
lda_prob<-predict(lda_fit,test_data,type='prob')
lda.roc<-multiclass.roc(test_data$playlist_genre,lda_prob$posterior)
lda.roc
mn_prob<-predict(mult.model,test_data,type='prob')
mn.roc<-multiclass.roc(test_data$playlist_genre,mn_prob)
```

```
mn.roc
rf.predict.prob<-predict(rf.fit,test_data,type='prob')
rf.predict.prob
rf.roc<-multiclass.roc(test_data$playlist_genre,rf.predict.prob)
rf.roc
bag.predict.prob<-predict(bag.data,test_data,type='prob')
bag.roc<-multiclass.roc(test_data$playlist_genre,bag.predict.prob)
bag.roc
```

A.5 Regression Methods

```
      (Intercept)       danceability             energy                key
      46.56473285         1.51786588         -5.28564326         0.13012522
         loudness               mode         speechiness       acousticness
       4.78350133         0.11252849         -0.22846619         0.62498493
 instrumentalness           liveness            valence              tempo
      -2.06112278         -0.50482673         -0.16328546         0.63212459
      duration_ms               year   playlist_genreedm playlist_genrelatin
      -2.68242998         0.23878935         -9.68686535        -1.46563684
`playlist_genrer&b`  playlist_genrerap  playlist_genrerock
      -6.84902271         -5.36397645         -0.01986216
```

Figure A.16: Best Subset Regression Coefficient

```
                              s0
(Intercept)         43.05856737
danceability         1.53703790
energy              -4.93515018
key                  0.04641964
loudness             4.43756099
mode                 0.22406107
speechiness         -0.27488640
acousticness         0.58132393
instrumentalness    -1.98767630
liveness            -0.33744966
valence             -0.13798730
tempo                0.58395414
duration_ms         -2.71529495
year                 0.59286289
playlist_genreedm   -6.52020000
playlist_genrelatin  1.84998697
playlist_genrepop    3.46116193
`playlist_genrer&b` -2.96686991
playlist_genrerap   -2.04032653
playlist_genrerock   3.50345589
```

Figure A.17: Ridge Regression Coefficient

```
#best subset fwd
library(leaps)
reg_fwd = regsubsets(track_popularity ~ ., data = data_process,
                     nvmax=19,method='forward')

names(summary(reg_fwd))
reg_summary=summary(reg_fwd)
```

```
                              s0
(Intercept)         43.51782182
danceability         1.53559772
energy              -5.18944983
key                  0.03935095
loudness             4.65570207
mode                 0.21245531
speechiness         -0.26047065
acousticness         0.52864658
instrumentalness    -1.95613298
liveness            -0.31538294
valence             -0.11122762
tempo                0.58980867
duration_ms         -2.74039803
year                 0.54599415
playlist_genreedm   -6.98060844
playlist_genrelatin  1.40172097
playlist_genrepop    3.02614897
`playlist_genrer&b` -3.47841553
playlist_genrerap   -2.54802807
playlist_genrerock   3.12520475
```

Figure A.18: Lasso Regression Coefficient

```
par(mfrow=c(2,2))

plot(reg_summary$rss, xlab="Number of Variables", ylab="RSS",type='l')
(best_model = which.min(reg_summary$rss))
points(best_model, reg_summary$rss[best_model], col="red", cex=2, pch=20)


plot(reg_summary$adjr2, xlab="Number of Variables", ylab="Adjusted RSq",
     type="l")

(best_model = which.max(reg_summary$adjr2))
points(best_model, reg_summary$adjr2[best_model], col="red", cex=2, pch=20)

plot(reg_summary$cp, xlab="Number of Variables", ylab="Cp", type="l")
(best_model = which.min(reg_summary$cp))
points(best_model, reg_summary$cp[best_model], col="red", cex=2, pch=20)

plot(reg_summary$bic, xlab="Number of Variables", ylab="BIC", type="l")
(best_model = which.min(reg_summary$bic))
points(best_model, reg_summary$bic[best_model], col="red", cex=2, pch=20)
coef(reg_fwd,18)

#Ridge regression
library(glmnet)
x_train=model.matrix(track_popularity~.,data=train)[,-1]
y_train=train$track_popularity
x_test=model.matrix(track_popularity~.,data=test)[,-1]
y_test=test$track_popularity
```

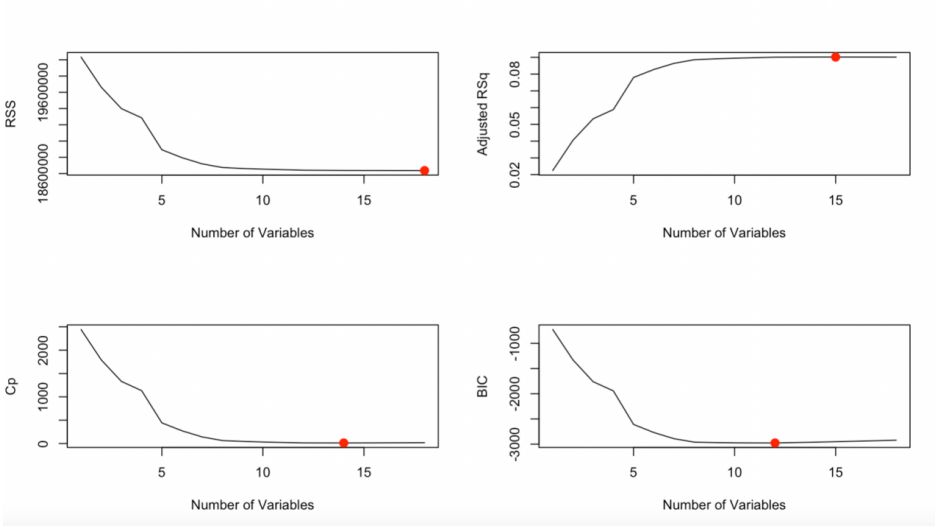

Figure A.19: Analysis for Best Subset Regression

```
cv.ridge <-cv.glmnet(x_train , y_train , alpha=0)
plot(cv.ridge)
coef(cv.ridge)
coef(glmnet(x_train ,y_train ,alpha=0, lambda=cv.ridge$lambda.min))
cv.ridge$lambda.min

ridge.train=glmnet(x_train ,y_train ,alpha=0,lambda = cv.ridge$lambda.min)
ridge.pre=predict(ridge.train ,newx =x_test)
mean((ridge.pre-y_test)^2)
rmse <-sqrt(apply((ridge.pre-y_test)^2,2,mean))
#lasso
lasso.fit=glmnet(x_train , y_train , alpha=1)
summary(lasso.fit)

cv.lasso=cv.glmnet(x_train ,y_train ,alpha=1)
plot(cv.lasso)
cv.lasso$lambda.min
coef(glmnet(x_train ,y_train ,alpha=1, lambda=cv.lasso$lambda.min))
lasso.train=glmnet(x_train ,y_train ,alpha = 1,lambda =cv.lasso$lambda.min )
lasso.pre=predict(lasso.train ,newx=x_test)
mean((lasso.pre-y_test)^2 )
```

Appendix B

Estimation of graphical models using lasso-related approaches

B.1 Node wise lasso

B.1.1 Node wise lasso 1 and 2 functions

```
#################################################
################################### NODE-WISE LASSO
#################################################

library('matlab')
library('gdata')
library('matrixcalc')
library('MASS')
library('pracma')
library('glasso')
library('network')
library('CVglasso')
library('glmnet')


#################################################
#### Linear Regression, without shrinking any coefficient


linear_regression=function(X, j){
  fit=lm(X[,j]~ X[,-j], data= as.data.frame(X))
  summary(fit)
}


#################################################
```

```
#################################################
#################################################
############# LASSO



# LASSO but without applying any validation method:
linear_regression_lasso=function(X,j){
  glmnet(X[,-j],X[,j])
}

# Plotting the coeff with respect to lambda and deviance:
plot_lasso=function(X,j){
  fit.lasso=glmnet(X[,-j],X[,j])
  par(mfrow=c(1,2))
  plot(fit.lasso, xvar="lambda", label= TRUE)
  plot(fit.lasso, xvar="dev", label= TRUE)
}

#########################################################
#########################################################
##### CROSS VALIDATION LASSO

##### Cross validation of k folds
cv_lasso=function(X,j,k){
  cv.glmnet(X[,-j],X[,j],nfolds = k)
}


##### Cross validation of k folds plot MSE with respect to Log(lambda):
cv_lasso_plot=function(X,j,k){
  cv.lasso= cv.glmnet(X[,-j],X[,j],nfolds = k)
  plot(cv.lasso)
}


################################## 1 STANDARD ERROR RULE
#########################################

##### Coefficient vector following the 1 standard error
rule using Cross validation of k folds
cv_lasso_standard_error_rule= function(X ,j, k){
  cv.lasso= cv.glmnet(X[,-j],X[,j],nfolds = k)
  coef(cv.lasso)

}

#### lambda following the 1 standard error rule
using Cross validation of k folds
```

```
cv_lasso_standard_error_rule_get_lambda= function(X ,j , k){
  cv.lasso= cv.glmnet(X[,-j],X[,j],nfolds = k)
  cv.lasso$lambda.1se
}


### Returns the estimated set E_1 (the pairs)
for the 1 standard error rule using CV with k folds
E_1_cv_lasso_standard_error_rule= function(X,k){
  E=list()
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = cv_lasso_standard_error_rule(X,i,k)
        model_j = cv_lasso_standard_error_rule(X,j,k)
        if (model_i[j+1,1]!=0 & model_j[i,1]!=0){
          E = c(E,list(c(i,j)))
        }
      }

    }

  }
  return(E)
}


### Returns the matrix of the estimated set E_1,
where if beta_ij != 0 and beta_ji != 0 then [i,j]=[j,i]= 1,
[i,j]= 0 else
E_1_cv_lasso_standard_error_rule_matrix= function(X,k){
  E=matrix(0,length(X[1,]),length(X[1,]))
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = cv_lasso_standard_error_rule(X,i,k)
        model_j = cv_lasso_standard_error_rule(X,j,k)
        if (model_i[j+1,1]!=0 & model_j[i,1]!=0){
          E[i,j]=1
          E[j,i]=1
        }
        else {
          E[i,j]=0
          E[j,i]=0
        }
      }

    }
```

```
  }
  return(E)
}

### Returns the estimated set E_2 (the pairs) for
the 1 standard error rule using CV with k folds
E_2_cv_lasso_standard_error_rule= function(X,k){
  E=list()
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = cv_lasso_standard_error_rule(X,i,k)
        model_j = cv_lasso_standard_error_rule(X,j,k)
        if (model_i[j+1,1]!=0 | model_j[i,1]!=0){
          E = c(E,list(c(i,j)))
        }
      }

    }

  }
  return(E)
}



### Returns the matrix of the estimated set E_2,
where if beta_ij != 0 or beta_ji!= 0 then [i,j]=[j,i]= 1,
[i,j]= 0 else
E_2_cv_lasso_standard_error_rule_matrix= function(X,k){
  E=matrix(0,length(X[1,]),length(X[1,]))
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = cv_lasso_standard_error_rule(X,i,k)
        model_j = cv_lasso_standard_error_rule(X,j,k)
        if (model_i[j+1,1]!=0 | model_j[i,1]!=0){
          E[i,j]=1
          E[j,i]=1
        }
        else {
          E[i,j]=0
          E[j,i]=0
        }
      }

    }
```

```
  }
  return(E)
}


######################################## BEST LAMBDA ######

##### Coefficient vector corresponding to the lowest
MSE using the best lambda using Cross Validation of k folds
cv_lasso_best_lambda= function(X,j,k){
  lambda = cv.glmnet(X[,-j],X[,j],nfolds = k)$lambda.min
  cv.lasso=glmnet(X[,-j],X[,j], lambda=lambda)
  coef(cv.lasso)
}

cv_lasso_best_lambda_get_lambda= function(X,j,k){
  lambda = cv.glmnet(X[,-j],X[,j],nfolds = k)$lambda.min
  return(lambda)
}

### Returns the estimated set E_1 (the pairs) for
the 1 standard error rule using CV with k folds
E_1_cv_lasso_best_lambda= function(X,k){
  E=list()
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = cv_lasso_best_lambda(X,i,k)
        model_j = cv_lasso_best_lambda(X,j,k)
        if (model_i[j+1,1]!=0 & model_j[i,1]!=0){
          E = c(E,list(c(i,j)))
        }
      }

    }

  }
  return(E)
}


### Returns the matrix of the estimated set E_1,
where if beta_ij != 0 and beta_ji!= 0 then [i,j]=[j,i]= 1,
[i,j]= 0 else
E_1_cv_lasso_best_lambda_matrix= function(X,k){
  E=matrix(0,length(X[1,]),length(X[1,]))
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
```

```
      model_i = cv_lasso_best_lambda(X,i,k)
      model_j = cv_lasso_best_lambda(X,j,k)
      if (model_i[j+1,1]!=0 & model_j[i,1]!=0){
        E[i,j]=1
        E[j,i]=1
      }
      else {
        E[i,j]=0
        E[j,i]=0
      }
    }

  }
  }
  return(E)
}


### Returns the estimated set E_2 (the pairs) for
the 1 standard error rule using CV with k folds
E_2_cv_lasso_best_lambda= function(X,k){
  E=list()
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = cv_lasso_best_lambda(X,i,k)
        model_j = cv_lasso_best_lambda(X,j,k)
        if (model_i[j+1,1]!=0 | model_j[i,1]!=0){
          E = c(E,list(c(i,j)))
        }
      }

    }

  }
  return(E)
}



### Returns the matrix of the estimated set E_2, where
if beta_ij != 0 or beta_ji != 0 then [i,j]=[j,i]= 1,
[i,j]= 0 else
E_2_cv_lasso_best_lambda_matrix= function(X,k){
  E=matrix(0,length(X[1,]),length(X[1,]))
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
```

```
        model_i = cv_lasso_best_lambda(X,i,k)
        model_j = cv_lasso_best_lambda(X,j,k)
        if (model_i[j+1,1]!=0 | model_j[i,1]!=0){
          E[i,j]=1
          E[j,i]=1
        }
        else {
          E[i,j]=0
          E[j,i]=0
        }
      }

    }

  }
  return(E)
}
```

```
#####################################################
#####################################################
#####################################################
#AIC APPROACH TO CHOOSE LAMBDA


##### Coefficient vector following the lambda which
minimises the AIC
lasso_AIC=function(X,j){
  fit=glmnet(X[,-j],X[,j])
  L = fit$nulldev - deviance(fit)
  d = fit$df
  aic <- -L+2*d
  AIC_lambda=fit$lambda[which.min(aic)]
  model=glmnet(X[,-j],X[,j],lambda=AIC_lambda)
  coef(model)
}

##### Coefficient vector following the lambda 1SE RULE
lasso_AIC_SE=function(X,j){
  fit=glmnet(X[,-j],X[,j])
  L = fit$nulldev - deviance(fit)
  d = fit$df
  aic = -L+2*d
  SE_AIC = sd(aic)/sqrt(length(aic))
  AIC_lambda_se = max(fit$lambda[which(aic<=min(aic)+SE_AIC)])
```

```
  model=glmnet(X[,-j],X[,j],lambda=AIC_lambda_se)
  coef(model)
}

##### get lambda which minimises the AIC
lasso_AIC_get_lambda=function(X,j){
  fit=glmnet(X[,-j],X[,j])
  L = fit$nulldev - deviance(fit)
  d = fit$df
  aic = -L+2*d
  AIC_lambda=fit$lambda[which.min(aic)]
  return(AIC_lambda)
}

##### get lambda wITH AIC SE rule
lasso_AIC_SE_get_lambda=function(X,j){
  fit=glmnet(X[,-j],X[,j])
  L = fit$nulldev - deviance(fit)
  d = fit$df
  aic = -L+2*d
  SE_AIC = sd(aic)/sqrt(length(aic))
  AIC_lambda_se = max(fit$lambda[which(aic<=min(aic)+SE_AIC)])
  return(AIC_lambda_se)
}

### Returns the estimated set E_1 (the pairs) for the AIC criterion
E_1_lasso_AIC= function(X){
  E=list()
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_AIC(X,i)
        model_j = lasso_AIC(X,j)
        if (model_i[j+1,1]!=0 & model_j[i,1]!=0){
          E = c(E,list(c(i,j)))
        }
      }

    }

  }
  return(E)
}



### Returns the matrix of the estimated set E_1, where
if beta_ij != 0 and beta_ji!= 0 then [i,j]=[j,i]= 1,
[i,j]= 0 else
```

```
E_1_lasso_AIC_matrix= function(X){
  E=matrix(0,length(X[1,]),length(X[1,]))
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_AIC(X,i)
        model_j = lasso_AIC(X,j)
        if (model_i[j+1,1]!=0 & model_j[i,1]!=0){
          E[i,j]=1
          E[j,i]=1
        }
        else {
          E[i,j]=0
          E[j,i]=0
        }
      }

    }

  }
  return(E)
}


### Returns the estimated set E_1 (the pairs) for
the AIC criterion SE rule
E_1_lasso_AIC_SE= function(X){
  E=list()
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_AIC_SE(X,i)
        model_j = lasso_AIC_SE(X,j)
        if (model_i[j+1,1]!=0 & model_j[i,1]!=0){
          E = c(E,list(c(i,j)))
        }
      }

    }

  }
  return(E)
}



### Returns the matrix of the estimated set E_1,
where if beta_ij != 0 and beta_ji != 0
then [i,j]=[j,i]= 1,  [i,j]= 0 else
```

```
E_1_lasso_AIC_SE_matrix= function(X){
  E=matrix(0,length(X[1,]),length(X[1,]))
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_AIC_SE(X,i)
        model_j = lasso_AIC_SE(X,j)
        if (model_i[j+1,1]!=0 & model_j[i,1]!=0){
          E[i,j]=1
          E[j,i]=1
        }
        else {
          E[i,j]=0
          E[j,i]=0
        }
      }

    }

  }
  return(E)
}

### Returns the estimated set E_2 (the pairs)
for the AIC criterion
E_2_lasso_AIC= function(X){
  E=list()
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_AIC(X,i)
        model_j = lasso_AIC(X,j)
        if (model_i[j+1,1]!=0 | model_j[i,1]!=0){
          E = c(E,list(c(i,j)))
        }
      }

    }

  }
  return(E)
}



### Returns the matrix of the estimated set E_1,
where if beta_ij != 0 and beta_ji != 0
then [i,j]=[j,i]= 1,  [i,j]= 0 else
E_2_lasso_AIC_matrix= function(X){
```

```
  E=matrix(0,length(X[1,]),length(X[1,]))
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_AIC(X,i)
        model_j = lasso_AIC(X,j)
        if (model_i[j+1,1]!=0 | model_j[i,1]!=0){
          E[i,j]=1
          E[j,i]=1
        }
        else {
          E[i,j]=0
          E[j,i]=0
        }
      }

    }

  }
  return(E)
}

### Returns the estimated set E_2 (the pairs)
for the AIC criterion SE rule
E_2_lasso_AIC_SE= function(X){
  E=list()
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_AIC_SE(X,i)
        model_j = lasso_AIC_sE(X,j)
        if (model_i[j+1,1]!=0 | model_j[i,1]!=0){
          E = c(E,list(c(i,j)))
        }
      }

    }

  }
  return(E)
}



### Returns the matrix of the estimated set E_1,
where if beta_ij != 0 and beta_ji != 0
then [i,j]=[j,i]= 1,  [i,j]= 0 else
E_2_lasso_AIC_SE_matrix= function(X){
  E=matrix(0,length(X[1,]),length(X[1,]))
```

```
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_AIC_SE(X,i)
        model_j = lasso_AIC_SE(X,j)
        if (model_i[j+1,1]!=0 | model_j[i,1]!=0){
          E[i,j]=1
          E[j,i]=1
        }
        else {
          E[i,j]=0
          E[j,i]=0
        }
      }

    }

  }
  return(E)
}



#####################################################
#####################################################
#####################################################
#BIC APPROACH TO CHOOSE LAMBDA


##### Coefficient vector following the lambda
which minimises the BIC
lasso_BIC=function(X,j){
  fit=glmnet(X[,-j],X[,j])
  L <- fit$nulldev - deviance(fit)
  d <- fit$df
  n <- fit$nobs
  BIC<-log(n)*d - L
  BIC_lambda=fit$lambda[which.min(BIC)]
  model=glmnet(X[,-j],X[,j],lambda=BIC_lambda)
  coef(model)
}


##### Get lambda which minimises the BIC
lasso_BIC_get_lambda=function(X,j){
  fit=glmnet(X[,-j],X[,j])
  L <- fit$nulldev - deviance(fit)
  d <- fit$df
  n <- fit$nobs
```

```
  BIC<-log(n)*d - L
  BIC_lambda=fit$lambda[which.min(BIC)]
  return(BIC_lambda)
}



##### Coefficient vector of BIC 1SE RULE
lasso_BIC_SE=function(X,j){
  fit=glmnet(X[,-j],X[,j])
  L <- fit$nulldev - deviance(fit)
  d <- fit$df
  n <- fit$nobs
  BIC<-log(n)*d - L
  SE_BIC = sd(BIC)/sqrt(length(BIC))
  BIC_lambda_se = max(fit$lambda[which(BIC<=min(BIC)+SE_BIC)])
  model=glmnet(X[,-j],X[,j],lambda=BIC_lambda_se)
  coef(model)
}


##### Get lambda which minimises the BIC 1SE RULE
lasso_BIC_SE_get_lambda=function(X,j){
  fit=glmnet(X[,-j],X[,j])
  L <- fit$nulldev - deviance(fit)
  d <- fit$df
  n <- fit$nobs
  SE_BIC = sd(BIC)/sqrt(length(BIC))
  BIC_lambda_se = max(fit$lambda[which(BIC<=min(BIC)+SE_BIC)])
  return(BIC_lambda_se)
}

### Returns the estimated set E_2 (the pairs) for the BIC criterion
E_1_lasso_BIC= function(X){
  E=list()
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_BIC(X,i)
        model_j = lasso_BIC(X,j)
        if (model_i[j+1,1]!=0 & model_j[i,1]!=0){
          E = c(E,list(c(i,j)))
        }
      }

    }

  }
  return(E)
```

```
}



### Returns the matrix of the estimated set E_1,
where if beta_ij != 0 and beta_ji != 0
then [i,j]=[j,i]= 1,  [i,j]= 0 else
E_1_lasso_BIC_matrix= function(X){
  E=matrix(0,length(X[1,]),length(X[1,]))
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_BIC(X,i)
        model_j = lasso_BIC(X,j)
        if (model_i[j+1,1]!=0 & model_j[i,1]!=0){
          E[i,j]=1
          E[j,i]=1
        }
        else {
          E[i,j]=0
          E[j,i]=0
        }
      }

    }

  }
  return(E)
}

### Returns the estimated set E_2 (the pairs)
for the BIC criterion 1 SE RULE
E_1_lasso_BIC_SE= function(X){
  E=list()
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_BIC_SE(X,i)
        model_j = lasso_BIC_SE(X,j)
        if (model_i[j+1,1]!=0 & model_j[i,1]!=0){
          E = c(E,list(c(i,j)))
        }
      }

    }

  }
  return(E)
}
```

```
### Returns the matrix of the estimated set E_1,
where if beta_ij != 0 and beta_ji != 0
then [i,j]=[j,i]= 1,  [i,j]= 0 else
E_1_lasso_BIC_SE_matrix= function(X){
  E=matrix(0,length(X[1,]),length(X[1,]))
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_BIC_SE(X,i)
        model_j = lasso_BIC_SE(X,j)
        if (model_i[j+1,1]!=0 & model_j[i,1]!=0){
          E[i,j]=1
          E[j,i]=1
        }
        else {
          E[i,j]=0
          E[j,i]=0
        }
      }

    }

  }
  return(E)
}

### Returns the estimated set E_2 (the pairs)
for the BIC criterion
E_2_lasso_BIC= function(X){
  E=list()
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_BIC(X,i)
        model_j = lasso_BIC(X,j)
        if (model_i[j+1,1]!=0 | model_j[i,1]!=0){
          E = c(E,list(c(i,j)))
        }
      }

    }

  }
  return(E)
}
```

```
### Returns the matrix of the estimated set E_1,
where if beta_ij != 0 and beta_ji != 0
then [i,j]=[j,i]= 1,  [i,j]= 0 else
E_2_lasso_BIC_matrix= function(X){
  E=matrix(0,length(X[1,]),length(X[1,]))
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_BIC(X,i)
        model_j = lasso_BIC(X,j)
        if (model_i[j+1,1]!=0 | model_j[i,1]!=0){
          E[i,j]=1
          E[j,i]=1
        }
        else {
          E[i,j]=0
          E[j,i]=0
        }
      }

    }

  }
  return(E)
}

### Returns the estimated set E_2 (the pairs)
for the BIC criterion 1SE RULE
E_2_lasso_BIC_SE= function(X){
  E=list()
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_BIC_SE(X,i)
        model_j = lasso_BIC_SE(X,j)
        if (model_i[j+1,1]!=0 | model_j[i,1]!=0){
          E = c(E,list(c(i,j)))
        }
      }

    }

  }
  return(E)
}
```

```
### Returns the matrix of the estimated set E_1,
where if beta_ij != 0 and beta_ji != 0
then [i,j]=[j,i]= 1,  [i,j]= 0 else
E_2_lasso_BIC_SE_matrix= function(X){
  E=matrix(0,length(X[1,]),length(X[1,]))
  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_BIC_SE(X,i)
        model_j = lasso_BIC_SE(X,j)
        if (model_i[j+1,1]!=0 | model_j[i,1]!=0){
          E[i,j]=1
          E[j,i]=1
        }
        else {
          E[i,j]=0
          E[j,i]=0
        }
      }

    }

  }
  return(E)
}
```

B.1.2 Node wise lasso 1 and 2 simulations

```
#########################################################
#########################################################
#########################################################
# SIMULATIONS - WITH COMPARISON OF ALL METHODS FOR E1


### perform simulation of specific n,p,delta,q combo
for E2 and give metrics as output
simulations_E1 = function(n,p,delta,q){

  #### Generate theta
a = sample(c(0, 0.5), size = (p*(p-1))/2, replace = TRUE, prob = c(q,1-q))
b = zeros(p)
lowerTriangle(b,byrow = TRUE ) <- a
upperTriangle(b) = lowerTriangle(b, byrow=TRUE) #for symmetry
diag(b)=1*delta
theta = b
is.positive.definite(theta, tol=1e-8)
#convert covariance matrix to correlation matrix
theta = cov2cor(theta)
is.positive.definite(theta, tol=1e-8)
sigma = inv(theta) #covariance matrix
trueA <- ifelse(lowerTriangle(theta)!=0,1,0)# classed of theta
theta_pairs_matrix = ifelse((theta)!=0,1,0) # pairs matrix of theta
NP = sum(sum(trueA == 1)) # No. of Positive
NN = sum(sum(trueA == 0)) # No. of Negative

#### Getting the set true pairs from theta:
true_pairs=function(theta){
  true=list()
  for (i in 1:p){
    for (j in 1:p){
      if (i>j & theta[i,j]!=0){
        true = c(true,list(c(i,j)))
      }

    }

  }
  return(true)
}

true_pairs(theta)

#########################################################################
### Simulating X
X=mvrnorm(n, mu=rep(0,p), Sigma = sigma)
```

```
#############################################################
#############################################################
METHODS

########## K FOLD
K_FOLD_E1 = function(X,k){

  #1-standard error rule lambda:


  A=E_1_cv_lasso_standard_error_rule_matrix(X ,k) # "Estimated matrix"
  estA_E1_lasso_standard_error_rule=ifelse(lowerTriangle(A)!=0,1,0)
  # Estimated classes

  TP_1SE_E1 = sum(sum(trueA == 1 & estA_E1_lasso_standard_error_rule == 1))
  FP_1SE_E1 = sum(sum(trueA == 0 & estA_E1_lasso_standard_error_rule == 1))

  TPR_1SE_E1 = TP_1SE_E1 / NP # True Positive Rate (Sensitivity/recall)
  FPR_1SE_E1= FP_1SE_E1 / NN # False Positive Rate

  precision_1SE_E1 = TP_1SE_E1/ (TP_1SE_E1+FP_1SE_E1)
  gmean_1SE_E1 = sqrt(TPR_1SE_E1*(1-FPR_1SE_E1))
  f_score_1SE_E1 = (2*(precision_1SE_E1)*(TPR_1SE_E1))/
  (precision_1SE_E1+TPR_1SE_E1)

  #Best-lambda

  B=E_1_cv_lasso_best_lambda_matrix(X,k) # "Estimated matrix"
  estA_E1_lasso_best_lambda=ifelse(lowerTriangle(B)!=0,1,0)
  # Estimated classes

  TP_BL_E1 = sum(sum(trueA == 1 & estA_E1_lasso_best_lambda == 1))
  FP_BL_E1 = sum(sum(trueA == 0 & estA_E1_lasso_best_lambda == 1))

  TPR_BL_E1 = TP_BL_E1 / NP # True Positive Rate (Sensitivity/recall)
  FPR_BL_E1 = FP_BL_E1 / NN # False Positive Rate

  precision_BL_E1 = TP_BL_E1/ (TP_BL_E1+FP_BL_E1)
  gmean_BL_E1 = sqrt(TPR_BL_E1*(1-FPR_BL_E1))
  f_score_BL_E1 = (2*(precision_BL_E1)*(TPR_BL_E1))/
  (precision_BL_E1+TPR_BL_E1)


  out_1SE_E1=c(TPR_1SE_E1,FPR_1SE_E1,precision_1SE_E1,
  gmean_1SE_E1,f_score_1SE_E1)
  out_BL_E1=c(TPR_BL_E1,FPR_BL_E1,precision_BL_E1,
  gmean_BL_E1,f_score_BL_E1)
```

```
  out_E1=matrix(c(out_BL_E1,out_1SE_E1),nrow=2,ncol=5,byrow=TRUE)
  rownames(out_E1) <- c(paste(toString(k),"- fold CV (BL RULE) (E1)"),
  paste(toString(k),"- fold CV (SE RULE) (E1)"))
  colnames(out_E1) <- c('Sensitivity','FPR','Precision','G - mean','F-Score')
  tab <- as.table(out_E1)

  par(mfrow=c(2,3))
  plot(network(B),label=1:p,
  main=paste("Estimated using",toString(k),
  "- fold CV \n(BL RULE) (E1)"),mode = "circle")
  plot(network(A),label=1:p,
  main=paste("Estimated using",toString(k),
  "- fold CV \n(SE RULE) (E1)"),mode = "circle")
  plot(network(theta_pairs_matrix),label=1:p,
  main="True network",mode = "circle")

  return(tab)
}

###5 fold:

t1=K_FOLD_E1(X,5)

## 10 fold:

t2=K_FOLD_E1(X,10)


######## AIC

#Best lambda
estA_E1_lasso_AIC=ifelse(lowerTriangle(E_1_lasso_AIC_matrix(X))!=0,1,0)
# Estimated classes
TP_E1_AIC = sum(sum(trueA == 1 & estA_E1_lasso_AIC== 1))
FP_E1_AIC = sum(sum(trueA == 0 & estA_E1_lasso_AIC == 1))

TPR_E1_AIC = TP_E1_AIC / NP # True Positive Rate (Sensitivity/recall)
FPR_E1_AIC= FP_E1_AIC / NN # False Positive Rate

precision_E1_AIC = TP_E1_AIC/ (TP_E1_AIC+FP_E1_AIC)
gmean_E1_AIC = sqrt(TPR_E1_AIC*(1-FPR_E1_AIC))
f_score_AIC = (2*(precision_E1_AIC)*(TPR_E1_AIC))/
(precision_E1_AIC+TPR_E1_AIC)


out_E1_AIC=matrix(c(TPR_E1_AIC,FPR_E1_AIC,precision_E1_AIC,
gmean_E1_AIC,f_score_AIC),nrow=1,ncol=5,byrow=TRUE)
#first row is for 1se,seconf is for BL.
#First col is TPR,second col is FPR and so on
```

```
rownames(out_E1_AIC) <- c("Minimising AIC (BL RULE) (E1)")
colnames(out_E1_AIC) <- c('Sensitivity','FPR','Precision','G-
mean','F-score')
t3=as.table(out_E1_AIC)

#1se rule
#Estimated classes
estA_E1_lasso_AIC_SE=ifelse(lowerTriangle(E_1_lasso_AIC_SE_matrix(X))!=0,1,0)


TP_E1_AIC_SE = sum(sum(trueA == 1 & estA_E1_lasso_AIC_SE== 1))
FP_E1_AIC_SE = sum(sum(trueA == 0 & estA_E1_lasso_AIC_SE == 1))

TPR_E1_AIC_SE = TP_E1_AIC_SE / NP # True Positive Rate (Sensitivity/recall)
FPR_E1_AIC_SE= FP_E1_AIC_SE / NN # False Positive Rate

precision_E1_AIC_SE = TP_E1_AIC_SE/ (TP_E1_AIC_SE+FP_E1_AIC_SE)
gmean_E1_AIC_SE = sqrt(TPR_E1_AIC_SE*(1-FPR_E1_AIC_SE))
f_score_AIC_SE = (2*(precision_E1_AIC_SE)*(TPR_E1_AIC_SE))/
(precision_E1_AIC_SE+TPR_E1_AIC_SE)


out_E1_AIC_SE=matrix(c(TPR_E1_AIC_SE,FPR_E1_AIC_SE,precision_E1_AIC_SE,
gmean_E1_AIC_SE,f_score_AIC_SE),nrow=1,ncol=5,byrow=TRUE)
#first row is for 1se,seconf is for BL.
#First col is TPR,second col is FPR and so on
rownames(out_E1_AIC_SE) <- c("Minimising AIC (SE RULE) (E1)")
colnames(out_E1_AIC_SE) <- c('Sensitivity','FPR','Precision',
'G - mean','F-score')
t4=as.table(out_E1_AIC_SE)

#### BIC


estA_E1_lasso_BIC=ifelse(lowerTriangle(E_1_lasso_BIC_matrix(X))!=0,1,0)
# Estimated classes
TP_E1_BIC = sum(sum(trueA == 1 & estA_E1_lasso_BIC== 1))
FP_E1_BIC = sum(sum(trueA == 0 & estA_E1_lasso_BIC == 1))

TPR_E1_BIC = TP_E1_BIC / NP # True Positive Rate (Sensitivity/recall)
FPR_E1_BIC= FP_E1_BIC / NN # False Positive Rate

precision_E1_BIC = TP_E1_BIC/ (TP_E1_BIC+FP_E1_BIC)
gmean_E1_BIC = sqrt(TPR_E1_BIC*(1-FPR_E1_BIC))
f_score_BIC = (2*(precision_E1_BIC)*(TPR_E1_BIC))/
(precision_E1_BIC+TPR_E1_BIC)


out_E1_BIC=matrix(c(TPR_E1_BIC,FPR_E1_BIC,precision_E1_BIC,
```

```
gmean_E1_BIC,f_score_BIC),nrow=1,ncol=5,byrow=TRUE)
#first row is for 1se,seconf is for BL.
#First col is TPR,second col is FPR and so on
rownames(out_E1_BIC) <- c("Minimising BIC (BL RULE) (E1)")
colnames(out_E1_BIC) <- c('Sensitivity','FPR','Precision','G - mean','F-score')
t5=as.table(out_E1_BIC)

#1se rule
estA_E1_lasso_BIC_SE=ifelse(lowerTriangle(E_1_lasso_BIC_SE_matrix(X))!=0,1,0)
TP_E1_BIC_SE = sum(sum(trueA == 1 & estA_E1_lasso_BIC_SE== 1))
FP_E1_BIC_SE = sum(sum(trueA == 0 & estA_E1_lasso_BIC_SE == 1))

TPR_E1_BIC_SE = TP_E1_BIC_SE / NP # True Positive Rate (Sensitivity/recall)
FPR_E1_BIC_SE= FP_E1_BIC_SE / NN # False Positive Rate

precision_E1_BIC_SE = TP_E1_BIC_SE/ (TP_E1_BIC_SE+FP_E1_BIC_SE)
gmean_E1_BIC_SE = sqrt(TPR_E1_BIC_SE*(1-FPR_E1_BIC_SE))
f_score_BIC_SE = (2*(precision_E1_BIC_SE)*(TPR_E1_BIC_SE))/
(precision_E1_BIC_SE+TPR_E1_BIC_SE)


out_E1_BIC_SE=matrix(c(TPR_E1_BIC_SE,FPR_E1_BIC_SE,precision_E1_BIC_SE,
gmean_E1_BIC_SE,f_score_BIC_SE),nrow=1,ncol=5,byrow=TRUE)
#first row is for 1se,seconf is for BL.
#First col is TPR,second col is FPR and so on
rownames(out_E1_BIC_SE) <- c("Minimising BIC (SE RULE) (E1)")
colnames(out_E1_BIC_SE) <- c('Sensitivity','FPR','Precision',
'G - mean','F-score')
t6=as.table(out_E1_BIC_SE)

###### FINAL TABLE FOR E1
E1_METHODS_TABLE = rbind(t1,t2,t3,t4,t5,t6)


par(mfrow=c(2,3))
plot(network(E_1_lasso_AIC_matrix(X)),label=1:p,
main="Estimated using AIC (BL RULE) (E1)",mode = "circle")
plot(network(E_1_lasso_AIC_SE_matrix(X)),label=1:p,
main="Estimated using AIC (SE RULE) (E1)",mode = "circle")
plot(network(theta_pairs_matrix),label=1:p,main="True network",
mode = "circle")
plot(network(E_1_lasso_BIC_matrix(X)),label=1:p,main="Estimated
using BIC(BL RULE) (E1)",mode = "circle")
plot(network(E_1_lasso_BIC_SE_matrix(X)),label=1:p,main="Estimated
using BIC (SE RULE) (E1)",mode = "circle")
plot(network(theta_pairs_matrix),label=1:p,main="True network",
mode = "circle")

return(E1_METHODS_TABLE)
```

```
}


### perform simulation of specific n,p,delta,q
combo a specified number of times
and give average value of each
of the metrics as output
ntt_E1= function(n,p,delta,q,times){
  ar <- array(0, dim=c(8, 5, times))
  b=c()
  x=c()
  data_sens=c()
  data_fpr=c()

  for (t in c(1:times)){
    ar[,,t]=simulations_E1(n,p,delta,q)
  }

  for (i in 1:8){
    for (j in 1:5){
      for (k in 1:times){
        x=c(x,ar[i,j,k])
      }
    }
  }

  for (k in seq(from=1, to=(times*8*5), by=times)){
    b=c(b,mean(x[k:(k+times-1)]))
  }


  #SENSITIVITY BOX PLOT
  for (k in seq(from=1, to=(times*8*5), by=5*times)){
     data_sens=c(data_sens,c(x[k:(k+times-1)]))
  }
  data_sens=matrix(data_sens,nrow=times,ncol=8)
  df_sens=as.data.frame(data_sens)
  par(mfrow=c(1,1))
 # ggplot(data=df)+geom_boxplot()
  boxplot(df_sens,las=2,cex.axis=0.7,main="Sensitivity across E1 methods",
  ylab="SensitIvity",names= c("5-fold CV (BL)","5-fold CV (SE)",
  "10-fold CV (BL)","10-fold CV (SE)","AIC (BL)",
  "AIC (SE)","BIC (BL)","BIC (SE)"))

  #FPR BOX PLOT
  for (k in seq(from=1+times, to=(times*8*5), by=5*times)){
    data_fpr=c(data_fpr,c(x[k:(k+times-1)]))
```

```
  }
  data_fpr=matrix(data_fpr,nrow=times,ncol=8)
  df_fpr=as.data.frame(data_fpr)
  par(mfrow=c(1,1))
  # ggplot(data=df)+geom_boxplot()
  boxplot(df_fpr,las=2,cex.axis=0.7,main="FPR across E1 methods",
  ylab="FPR",names= c("5-fold CV (BL)","5-fold CV (SE)",
  "10-fold CV (BL)","10-fold CV (SE)","AIC (BL)",
  "AIC (SE)","BIC (BL)","BIC (SE)"))


  b=matrix(b,nrow=8,ncol=5,byrow = TRUE)
  rownames(b) <- c("5 - fold CV (BL RULE)
    (E1)","5 - fold CV (SE RULE)          (E1)",
    "10 - fold CV (BL RULE) (E1)",
    "10 - fold CV (SE RULE) (E1)","Minimising AIC (BL RULE) (E1)",
  "Minimising AIC (SE RULE) (E1)","Minimising BIC (BL RULE) (E1)",
  "Minimising BIC (SE RULE) (E1)")

  colnames(b) <- c('Sensitivity','FPR','Precision','G - mean',
  'F-Score')
  out=as.table(b)
  return(list(ar,out))

}

ntt_E1(500,10,4,0.5,50)


#################################################
#################################################
# SIMULATIONS - WITH COMPARISON OF ALL METHODS FOR E2


### perform simulation of specific n,p,delta,q combo for E2
and give metrics as output
simulations_E2 = function(n,p,delta,q){

#### Generate theta
  a = sample(c(0, 0.5), size = (p*(p-1))/2, replace = TRUE, prob = c(q,1-q))
  b = zeros(p)
  lowerTriangle(b,byrow = TRUE ) <- a
  upperTriangle(b) = lowerTriangle(b, byrow=TRUE) #for symmetry
  diag(b)=1*delta
  theta = b
  is.positive.definite(theta, tol=1e-8)
  #convert covariance matrix to correlation matrix
  theta = cov2cor(theta)
  is.positive.definite(theta, tol=1e-8)
```

```
sigma = inv(theta) #covariance matrix
trueA <- ifelse(lowerTriangle(theta)!=0,1,0)# classed of theta
theta_pairs_matrix = ifelse((theta)!=0,1,0) # pairs matrix of theta
NP = sum(sum(trueA == 1)) # No. of Positive
NN = sum(sum(trueA == 0)) # No. of Negative

#### Getting the set true pairs from theta:
true_pairs=function(theta){
  true=list()
  for (i in 1:p){
    for (j in 1:p){
      if (i>j & theta[i,j]!=0){
        true = c(true,list(c(i,j)))
      }

    }

  }
  return(true)
}

true_pairs(theta)

##########################################################
### Simulating X
X=mvrnorm(n, mu=rep(0,p), Sigma = sigma)

#########################################################################
#############################################################
METHODS

########## K FOLD
K_FOLD_E2 = function(X,k){

  #1-standard error rule lambda:


  A=E_2_cv_lasso_standard_error_rule_matrix(X ,k)
  # "Estimated matrix"
  estA_E2_lasso_standard_error_rule=ifelse(lowerTriangle(A)!=0,1,0)
  # Estimated classes

  TP_1SE_E2 = sum(sum(trueA == 1 & estA_E2_lasso_standard_error_rule == 1))
  FP_1SE_E2 = sum(sum(trueA == 0 & estA_E2_lasso_standard_error_rule == 1))

  TPR_1SE_E2 = TP_1SE_E2 / NP # True Positive Rate (Sensitivity/recall)
  FPR_1SE_E2= FP_1SE_E2 / NN # False Positive Rate

  precision_1SE_E2 = TP_1SE_E2/ (TP_1SE_E2+FP_1SE_E2)
```

```
  gmean_1SE_E2 = sqrt(TPR_1SE_E2*(1-FPR_1SE_E2))
  f_score_1SE_E2 = (2*(precision_1SE_E2)*(TPR_1SE_E2))/
  (precision_1SE_E2+TPR_1SE_E2)


  #Best-lambda

  B=E_2_cv_lasso_best_lambda_matrix(X,k) # "Estimated matrix"
  estA_E2_lasso_best_lambda=ifelse(lowerTriangle(B)!=0,1,0)
  # Estimated classes

  TP_BL_E2 = sum(sum(trueA == 1 & estA_E2_lasso_best_lambda == 1))
  FP_BL_E2 = sum(sum(trueA == 0 & estA_E2_lasso_best_lambda == 1))

  TPR_BL_E2 = TP_BL_E2 / NP # True Positive Rate (Sensitivity/recall)
  FPR_BL_E2 = FP_BL_E2 / NN # False Positive Rate

  precision_BL_E2 = TP_BL_E2/ (TP_BL_E2+FP_BL_E2)
  gmean_BL_E2 = sqrt(TPR_BL_E2*(1-FPR_BL_E2))
  f_score_BL_E2 = (2*(precision_BL_E2)*(TPR_BL_E2))/
  (precision_BL_E2+TPR_BL_E2)


  out_1SE_E2=c(TPR_1SE_E2,FPR_1SE_E2,precision_1SE_E2,
  gmean_1SE_E2,f_score_1SE_E2)
  out_BL_E2=c(TPR_BL_E2,FPR_BL_E2,precision_BL_E2,
  gmean_BL_E2,f_score_BL_E2)

  out_E2=matrix(c(out_BL_E2,out_1SE_E2),nrow=2,ncol=5,byrow=TRUE)
  rownames(out_E2) <- c(paste(toString(k),"- fold CV (BL RULE) (E2)"),
  paste(toString(k),"- fold CV (SE RULE) (E2)"))
  colnames(out_E2) <- c('Sensitivity','FPR','Precision',
  'G - mean','F-Score')
  tab <- as.table(out_E2)

  par(mfrow=c(2,3))
  plot(network(B),label=1:p
  ,main=paste("Estimated using",toString(k),
  "- fold CV \n(BL RULE) (E2)"),mode = "circle")
  plot(network(A),label=1:p,
  main=paste("Estimated using",toString(k),
  "- fold CV \n(SE RULE) (E2)"),mode = "circle")
  plot(network(theta_pairs_matrix),
  label=1:p,main="True network",mode = "circle")

  return(tab)
}

###5 fold:
```

```
t1=K_FOLD_E2(X,5)

## 10 fold:

t2=K_FOLD_E2(X,10)


######## AIC

#Best lambda
estA_E2_lasso_AIC=ifelse(lowerTriangle(E_2_lasso_AIC_matrix(X))!=0,1,0)
# Estimated classes
TP_E2_AIC = sum(sum(trueA == 1 & estA_E2_lasso_AIC== 1))
FP_E2_AIC = sum(sum(trueA == 0 & estA_E2_lasso_AIC == 1))

TPR_E2_AIC = TP_E2_AIC / NP # True Positive Rate (Sensitivity/recall)
FPR_E2_AIC= FP_E2_AIC / NN # False Positive Rate

precision_E2_AIC = TP_E2_AIC/ (TP_E2_AIC+FP_E2_AIC)
gmean_E2_AIC = sqrt(TPR_E2_AIC*(1-FPR_E2_AIC))
f_score_AIC = (2*(precision_E2_AIC)*(TPR_E2_AIC))/
(precision_E2_AIC+TPR_E2_AIC)


out_E2_AIC=matrix(c(TPR_E2_AIC,FPR_E2_AIC,precision_E2_AIC,
gmean_E2_AIC,f_score_AIC),nrow=1,ncol=5,byrow=TRUE)
#first row is for 1se,seconf is for BL.
#First col is TPR,second col is FPR and so on
rownames(out_E2_AIC) <- c("Minimising AIC (BL RULE) (E2)")
colnames(out_E2_AIC) <- c('Sensitivity','FPR','Precision',
'G - mean','F-score')
t3=as.table(out_E2_AIC)

#1se rule
estA_E2_lasso_AIC_SE=ifelse(lowerTriangle(E_2_lasso_AIC_SE_matrix(X))
!=0,1,0) # Estimated classes
TP_E2_AIC_SE = sum(sum(trueA == 1 & estA_E2_lasso_AIC_SE== 1))
FP_E2_AIC_SE = sum(sum(trueA == 0 & estA_E2_lasso_AIC_SE == 1))

TPR_E2_AIC_SE = TP_E2_AIC_SE / NP # True Positive Rate (Sensitivity/recall)
FPR_E2_AIC_SE= FP_E2_AIC_SE / NN # False Positive Rate

precision_E2_AIC_SE = TP_E2_AIC_SE/ (TP_E2_AIC_SE+FP_E2_AIC_SE)
gmean_E2_AIC_SE = sqrt(TPR_E2_AIC_SE*(1-FPR_E2_AIC_SE))
f_score_AIC_SE = (2*(precision_E2_AIC_SE)*(TPR_E2_AIC_SE))/
(precision_E2_AIC_SE+TPR_E2_AIC_SE)
```

```
out_E2_AIC_SE=matrix(c(TPR_E2_AIC_SE,FPR_E2_AIC_SE,
precision_E2_AIC_SE,
gmean_E2_AIC_SE,f_score_AIC_SE),
nrow=1,ncol=5,byrow=TRUE)
#first row is for 1se,seconf is for BL.
#First col is TPR,second col is FPR and so on
rownames(out_E2_AIC_SE) <- c("Minimising AIC (SE RULE) (E2)")
colnames(out_E2_AIC_SE) <- c('Sensitivity','FPR','Precision',
'G - mean','F-score')
t4=as.table(out_E2_AIC_SE)

#### BIC


estA_E2_lasso_BIC=ifelse(lowerTriangle(E_2_lasso_BIC_matrix(X))
!=0,1,0)
# Estimated classes
TP_E2_BIC = sum(sum(trueA == 1 & estA_E2_lasso_BIC== 1))
FP_E2_BIC = sum(sum(trueA == 0 & estA_E2_lasso_BIC == 1))

TPR_E2_BIC = TP_E2_BIC / NP # True Positive Rate (Sensitivity/recall)
FPR_E2_BIC= FP_E2_BIC / NN # False Positive Rate

precision_E2_BIC = TP_E2_BIC/ (TP_E2_BIC+FP_E2_BIC)
gmean_E2_BIC = sqrt(TPR_E2_BIC*(1-FPR_E2_BIC))
f_score_BIC = (2*(precision_E2_BIC)*(TPR_E2_BIC))/
(precision_E2_BIC+TPR_E2_BIC)


out_E2_BIC=matrix(c(TPR_E2_BIC,FPR_E2_BIC,precision_E2_BIC,
gmean_E2_BIC,f_score_BIC),nrow=1,ncol=5,byrow=TRUE)
#first row is for 1se,seconf is for BL.
#First col is TPR,second col is FPR and so on
rownames(out_E2_BIC) <- c("Minimising BIC (BL RULE) (E2)")
colnames(out_E2_BIC) <- c('Sensitivity','FPR','Precision',
'G - mean','F-score')
t5=as.table(out_E2_BIC)

#1se rule
estA_E2_lasso_BIC_SE=ifelse(lowerTriangle(E_2_lasso_BIC_SE_matrix(X))
!=0,1,0) # Estimated classes
TP_E2_BIC_SE = sum(sum(trueA == 1 & estA_E2_lasso_BIC_SE== 1))
FP_E2_BIC_SE = sum(sum(trueA == 0 & estA_E2_lasso_BIC_SE == 1))

TPR_E2_BIC_SE = TP_E2_BIC_SE / NP # True Positive Rate (Sensitivity/recall)
FPR_E2_BIC_SE= FP_E2_BIC_SE / NN # False Positive Rate

precision_E2_BIC_SE = TP_E2_BIC_SE/ (TP_E2_BIC_SE+FP_E2_BIC_SE)
gmean_E2_BIC_SE = sqrt(TPR_E2_BIC_SE*(1-FPR_E2_BIC_SE))
```

```
  f_score_BIC_SE = (2*(precision_E2_BIC_SE)*(TPR_E2_BIC_SE))/
  (precision_E2_BIC_SE+TPR_E2_BIC_SE)


  out_E2_BIC_SE=matrix(c(TPR_E2_BIC_SE,FPR_E2_BIC_SE,precision_E2_BIC_SE,
  gmean_E2_BIC_SE,f_score_BIC_SE),nrow=1,ncol=5,byrow=TRUE)
  #first row is for 1se,seconf is for BL.
  #First col is TPR,second col is FPR and so on
  rownames(out_E2_BIC_SE) <- c("Minimising BIC (SE RULE) (E2)")
  colnames(out_E2_BIC_SE) <- c('Sensitivity','FPR','Precision',
  'G - mean','F-score')
  t6=as.table(out_E2_BIC_SE)

  ###### FINAL TABLE FOR E2
  E2_METHODS_TABLE = rbind(t1,t2,t3,t4,t5,t6)


  par(mfrow=c(2,3))
  plot(network(E_2_lasso_AIC_matrix(X)),label=1:p,
  main="Estimated using AIC (BL RULE) (E2)",mode = "circle")
  plot(network(E_2_lasso_AIC_SE_matrix(X)),label=1:p
  ,main="Estimated using AIC (SE RULE) (E2)",mode = "circle")
  plot(network(theta_pairs_matrix),label=1:p,
  main="True network",mode = "circle")
  plot(network(E_2_lasso_BIC_matrix(X)),label=1:p,
  main="Estimated using BIC (BL RULE) (E2)",mode = "circle")
  plot(network(E_2_lasso_BIC_SE_matrix(X)),label=1:p,
  main="Estimated using BIC (SE RULE) (E2)",mode = "circle")
  plot(network(theta_pairs_matrix),label=1:p,main="True network",
  mode = "circle")

  return(E2_METHODS_TABLE)

}


### perform simulation of specific n,p,delta,q combo a
specified number of times and give average value of
each of the metrics as output
ntt_E2= function(n,p,delta,q,times){
  ar <- array(0, dim=c(8, 5, times))
  b=c()
  x=c()
  data_sens=c()
  data_fpr=c()

  for (t in c(1:times)){
    ar[,,t]=simulations_E2(n,p,delta,q)
  }
```

```
for (i in 1:8){
  for (j in 1:5){
    for (k in 1:times){
      x=c(x,ar[i,j,k])
    }
  }
}

for (k in seq(from=1, to=(times*8*5), by=times)){
  b=c(b,mean(x[k:(k+times-1)]))
}


#SENSITIVITY BOX PLOT
for (k in seq(from=1, to=(times*8*5), by=5*times)){
  data_sens=c(data_sens,c(x[k:(k+times-1)]))
}
data_sens=matrix(data_sens,nrow=times,ncol=8)
df_sens=as.data.frame(data_sens)
par(mfrow=c(1,1))
# ggplot(data=df)+geom_boxplot()
boxplot(df_sens,las=2,cex.axis=0.7,main="Sensitivity across E2 methods",
ylab="SensitIvity",names= c("5-fold CV (BL)","5-fold CV (SE)",
"10-fold CV (BL)","10-fold CV (SE)","AIC (BL)",
"AIC (SE)","BIC (BL)","BIC (SE)"))

#FPR BOX PLOT
for (k in seq(from=1+times, to=(times*8*5), by=5*times)){
  data_fpr=c(data_fpr,c(x[k:(k+times-1)]))
}
data_fpr=matrix(data_fpr,nrow=times,ncol=8)
df_fpr=as.data.frame(data_fpr)
par(mfrow=c(1,1))
boxplot(df_fpr,las=2,cex.axis=0.7,main="FPR across E2 methods",
ylab="FPR",names= c("5-fold CV (BL)","5-fold CV (SE)",
"10-fold CV (BL)","10-fold CV (SE)","AIC (BL)","AIC (SE)",
"BIC (BL)","BIC (SE)"))


b=matrix(b,nrow=8,ncol=5,byrow = TRUE)
rownames(b) <- c("5 - fold CV (BL RULE) (E2)",
"5 - fold CV (SE RULE) (E2)","10 - fold CV (BL RULE) (E2)",
"10 - fold CV (SE RULE) (E2)",
"Minimising AIC (BL RULE) (E2)","Minimising AIC (SE RULE) (E2)",
"Minimising BIC (BL RULE) (E2)","Minimising BIC (SE RULE) (E2)")
colnames(b) <- c('Sensitivity','FPR','Precision',
'G - mean',
'F-Score')
```

```
  out=as.table(b)
  return(list(ar,out))

}

ntt_E2(500,10,4,0.5,50)
```

B.2 Graphical Lasso

B.2.1 Graphical Lasso functions

```
#########################################
#################################
GRAPHICAL-WISE LASSO ###################
#############################################

library('matlab')
library('gdata')
library('matrixcalc')
library('MASS')
library('pracma')
library('glasso')
library('network')
library('CVglasso')
library('cglasso')


#########################################
#########################################
# CROSS VALIDATION ######################

# Outputs best lambda, glasso estimate of inverse covariance matrix
(theta), avergae cv error for each lambda,
with best lambda from CV(minimising loglikelihood)
with k-folds:
glasso_cv_best=function(X,k,n){

  av_cv_error_each_lambda=c()
  CV = CVglasso(X,cov(X),K=k, crit.cv = 'loglik')
  cv_best_lambda=CV$Tuning[2]
  Cv_error_each_fold=CV$CV.error
  for (i in c(1:nrow(Cv_error_each_fold))){
     av_cv_error_each_lambda[i]=mean(Cv_error_each_fold[i,])
  }
  est=glasso(cov(X),rho = cv_best_lambda,n)$wi;
  plot(CV)
  return(list(cv_best_lambda,est,av_cv_error_each_lambda))
}
```

```
glasso_cv_1se=function(X,k,n){

  av_cv_error_each_lambda=c()
  CV = CVglasso(X,cov(X),K=k,crit.cv = 'loglik')

  Cv_error_each_fold=CV$CV.error
  for (i in c(1:nrow(Cv_error_each_fold))){
    av_cv_error_each_lambda[i]=mean(Cv_error_each_fold[i,])
  }

  SE_rmse=sd(av_cv_error_each_lambda)/sqrt(length(av_cv_error_each_lambda))
  # SE of lambda
  SE_lambda=max(CV$Lambdas
  [which(av_cv_error_each_lambda<=
  (min(av_cv_error_each_lambda)+SE_rmse))])
  # chosen lambda
  plot(CV)

  est=glasso(cov(X),rho = SE_lambda,n)$wi;
  return(list(SE_lambda,est))
}

#### Returns the estimated set E_3 (the pairs) using CV with k-folds (best):
E_3_cv_best=function(X,k,n,p){
  E=list()
  est = unlist(glasso_cv_best(X,k,n)[2])
  est=matrix(est,nrow=p,ncol=p)
  for (i in 1:p){
    for (j in 1:p){
      if (i>j & est[i,j]!=0){
        E = c(E,list(c(i,j)))
      }

    }

  }
  return(E)
}

#### Returns the matrix for the estimated set E_3 using CV (best):
E_3_cv_matrix_best = function(X,k,n,p){
  E=ifelse(unlist((glasso_cv_best(X,k,n)[2]))!=0,1,0)
  E=matrix(E,nrow=p,ncol=p)
  return(E)
}


#### Returns the estimated set E_3 (the pairs) using CV with k-folds (1se):
```

```
E_3_cv_1se=function(X,k,n,p){
  E=list()
  est = unlist(glasso_cv_1se(X,k,n)[2])
  est=matrix(est,nrow=p,ncol=p)
  for (i in 1:p){
    for (j in 1:p){
      if (i>j & est[i,j]!=0){
        E = c(E,list(c(i,j)))
      }

    }

  }
  return(E)
}

#### Returns the matrix for the estimated set E_3 using CV (1se):
E_3_cv_matrix_1se = function(X,k,n,p){
  E=ifelse(unlist((glasso_cv_1se(X,k,n)[2]))!=0,1,0)
  E=matrix(E,nrow=p,ncol=p)
  return(E)
}

########################################
################################## AIC
########################################

glasso_cv_best_AIC=function(X,n,p,theta){


  CV = CVglasso(X,cov(X),K=1,crit.cv = 'AIC')
  cv_best_lambda=CV$Tuning[2]
  Cv_error=CV$CV.error
  est=glasso(cov(X),rho = cv_best_lambda,n)$wi;
  mse = sqrt(apply(((upper.triangle(theta)-eye(p))-
  (upper.triangle(est)-eye(p)))^2,1,mean))

  return(list(cv_best_lambda,est,Cv_error,mse))
}

glasso_cv_1se_AIC=function(X,n){


  CV = CVglasso(X,cov(X),K=1,crit.cv = 'AIC')
  Cv_error=CV$CV.error
  SE_rmse=sd(Cv_error)/sqrt(length(Cv_error)) # SE of lambda
  SE_lambda=max(CV$Lambdas[which(Cv_error<=(min(Cv_error)+SE_rmse))])
  # chosen lambda
  est=glasso(cov(X),rho = SE_lambda,n)$wi;
```

```
  return(list(SE_lambda,est))
}

#### Returns the estimated set E_3 (the pairs) using CV with k-folds (best):
E_3_cv_AIC_best=function(X,n,p,theta){
  E=list()
  est = unlist(glasso_cv_best_AIC(X,n,p,theta)[2])
  est=matrix(est,nrow=p,ncol=p)
  for (i in 1:p){
    for (j in 1:p){
      if (i>j & est[i,j]!=0){
        E = c(E,list(c(i,j)))
      }

    }

  }
  return(E)
}

#### Returns the matrix for the estimated set E_3 using CV (best):
E_3_cv_AIC_matrix_best = function(X,n,p,theta){
  E=ifelse(unlist((glasso_cv_best_AIC(X,n,p,theta)[2]))!=0,1,0)
  E=matrix(E,nrow=p,ncol=p)
  return(E)
}

#### Returns the estimated set E_3 (the pairs) using CV with k-folds (1se):
E_3_cv_AIC_1se=function(X,n,p){
  E=list()
  est = unlist(glasso_cv_1se_AIC(X,n)[2])
  est=matrix(est,nrow=p,ncol=p)
  for (i in 1:p){
    for (j in 1:p){
      if (i>j & est[i,j]!=0){
        E = c(E,list(c(i,j)))
      }

    }

  }
  return(E)
}

#### Returns the matrix for the estimated set E_3 using CV(1se):
E_3_cv_AIC_matrix_1se = function(X,n,p){
  E=ifelse(unlist((glasso_cv_1se_AIC(X,n)[2]))!=0,1,0)
  E=matrix(E,nrow=p,ncol=p)
  return(E)
```

```
}

#####################################
################################### BIC
#####################################

glasso_cv_best_BIC=function(X,n,p,theta){

  CV = CVglasso(X,cov(X),K=1,crit.cv = 'BIC')
  cv_best_lambda=CV$Tuning[2]
  Cv_error=CV$CV.error
  est=glasso(cov(X),rho = cv_best_lambda,n)$wi;
  mse = sqrt(apply(((upper.triangle(theta)-eye(p))-
  (upper.triangle(est)-eye(p)))^2,1,mean))
  return(list(cv_best_lambda,est,Cv_error,mse))
}

glasso_cv_1se_BIC=function(X,n){

  CV = CVglasso(X,cov(X),K=1,crit.cv = 'BIC')
  Cv_error=CV$CV.error
  SE_rmse=sd(Cv_error)/sqrt(length(Cv_error)) # SE of lambda
  SE_lambda=max(CV$Lambdas[which(Cv_error<=(min(Cv_error)+SE_rmse))])
  # chosen lambda
  est=glasso(cov(X),rho = SE_lambda,n)$wi;
  return(list(SE_lambda,est))
}

#### Returns the estimated set E_3 (the pairs) using CV with k-folds (best):
E_3_cv_BIC_best=function(X,n,p,theta){
  E=list()
  est = unlist(glasso_cv_best_BIC(X,n,p,theta)[2])
  est=matrix(est,nrow=p,ncol=p)
  for (i in 1:p){
    for (j in 1:p){
      if (i>j & est[i,j]!=0){
        E = c(E,list(c(i,j)))
      }

    }

  }
  return(E)
}

#### Returns the matrix for the estimated set E_3 using CV (best):
E_3_cv_BIC_matrix_best = function(X,n,p,theta){
  E=ifelse(unlist((glasso_cv_best_BIC(X,n,p,theta)[2]))!=0,1,0)
  E=matrix(E,nrow=p,ncol=p)
```

```
  return(E)
}


#### Returns the estimated set E_3 (the pairs) using CV with k-folds (1se):
E_3_cv_BIC_1se=function(X,n,p){
  E=list()
  est = unlist(glasso_cv_1se_BIC(X,n)[2])
  est=matrix(est,nrow=p,ncol=p)
  for (i in 1:p){
    for (j in 1:p){
      if (i>j & est[i,j]!=0){
        E = c(E,list(c(i,j)))
      }

    }

  }
  return(E)
}

#### Returns the matrix for the estimated set E_3 using CV(1se):
E_3_cv_BIC_matrix_1se = function(X,n,p){
  E=ifelse(unlist((glasso_cv_1se_BIC(X,n)[2]))!=0,1,0)
  E=matrix(E,nrow=p,ncol=p)
  return(E)
}

############################################## IMPLEMENTATION


#####loglike
K_FOLD_E3 = function(X,k,n,p,trueA,NP,NN,theta_pairs_matrix){

  #best lambda
  A=E_3_cv_matrix_best(X,k,n,p) # "Estimated matrix"
  estA_E3_cv_best=ifelse(lowerTriangle(A)!=0,1,0)# Estimated classes

  TP_CV_E3_best = sum(sum(trueA == 1 & estA_E3_cv_best == 1))
  FP_CV_E3_best = sum(sum(trueA == 0 & estA_E3_cv_best == 1))

  TPR_CV_E3_best = TP_CV_E3_best / NP
  # True Positive Rate (Sensitivity/recall)
  FPR_CV_E3_best= FP_CV_E3_best / NN # False Positive Rate

  precision_CV_E3_best = TP_CV_E3_best/ (TP_CV_E3_best+FP_CV_E3_best)
  gmean_CV_E3_best = sqrt(TPR_CV_E3_best*(1-FPR_CV_E3_best))
  f_score_CV_E3_best = (2*(precision_CV_E3_best)*(TPR_CV_E3_best))/
  (precision_CV_E3_best+TPR_CV_E3_best)
```

```
out_CV_E3_best=matrix(c(TPR_CV_E3_best,FPR_CV_E3_best,
precision_CV_E3_best,gmean_CV_E3_best,f_score_CV_E3_best),
nrow=1,ncol=5,byrow=TRUE)
#first row is for 1se,seconf is for BL.
#First col is TPR,second col is FPR and so on
rownames(out_CV_E3_best) <- c(paste(toString(k),"- fold CV (E3)
(Best lambda)"))
colnames(out_CV_E3_best) <- c('Sensitivity','FPR','Precision',
'G - mean','F-score')
tab_best=as.table(out_CV_E3_best)

#1se
B=E_3_cv_matrix_1se(X,k,n,p) # "Estimated matrix"
estA_E3_cv_1se=ifelse(lowerTriangle(B)!=0,1,0)# Estimated classes

TP_CV_E3_1se = sum(sum(trueA == 1 & estA_E3_cv_1se == 1))
FP_CV_E3_1se = sum(sum(trueA == 0 & estA_E3_cv_1se == 1))

TPR_CV_E3_1se = TP_CV_E3_1se / NP # True Positive Rate (Sensitivity/recall)
FPR_CV_E3_1se= FP_CV_E3_1se / NN # False Positive Rate

precision_CV_E3_1se = TP_CV_E3_1se/ (TP_CV_E3_1se+FP_CV_E3_1se)
gmean_CV_E3_1se = sqrt(TPR_CV_E3_1se*(1-FPR_CV_E3_1se))
f_score_CV_E3_1se = (2*(precision_CV_E3_1se)*(TPR_CV_E3_1se))/
(precision_CV_E3_1se+TPR_CV_E3_1se)


out_CV_E3_1se=matrix(c(TPR_CV_E3_1se,FPR_CV_E3_1se,precision_CV_E3_1se,
gmean_CV_E3_1se,f_score_CV_E3_1se),nrow=1,ncol=5,byrow=TRUE)
#first row is for 1se,seconf is for BL.
#First col is TPR,second col is FPR and so on
rownames(out_CV_E3_1se) <- c(paste(toString(k),"- fold CV (E3) (1SE)"))
colnames(out_CV_E3_1se) <- c('Sensitivity','FPR','Precision',
'G - mean','F-score')
tab_1se=as.table(out_CV_E3_1se)

tab=rbind(tab_best,tab_1se)

par(mfrow=c(2,2))
plot(network(A),label=1:p,main=paste("Estimated using",
toString(k),"- fold CV \n (E3) (Best lambda)"),
mode = "circle")
plot(network(theta_pairs_matrix),label=1:p,
main="True network",mode = "circle")
plot(network(B),label=1:p,main=paste("Estimated using",toString(k),
"- fold CV \n (E3) (1SE)"),mode = "circle")
plot(network(theta_pairs_matrix),label=1:p,
```

```
  main="True network",mode = "circle")

  return(tab)
}

#####AIC
K_FOLD_E3_AIC = function(X,n,p,trueA,NP,NN,theta_pairs_matrix,theta){

  #best lambda
  A=E_3_cv_AIC_matrix_best(X,n,p,theta) # "Estimated matrix"
  estA_E3_cv_best=ifelse(lowerTriangle(A)!=0,1,0)# Estimated classes

  TP_CV_E3_best = sum(sum(trueA == 1 & estA_E3_cv_best == 1))
  FP_CV_E3_best = sum(sum(trueA == 0 & estA_E3_cv_best == 1))

  TPR_CV_E3_best = TP_CV_E3_best / NP # True Positive Rate (Sensitivity/recall)
  FPR_CV_E3_best= FP_CV_E3_best / NN # False Positive Rate

  precision_CV_E3_best = TP_CV_E3_best/ (TP_CV_E3_best+FP_CV_E3_best)
  gmean_CV_E3_best = sqrt(TPR_CV_E3_best*(1-FPR_CV_E3_best))
  f_score_CV_E3_best = (2*(precision_CV_E3_best)*(TPR_CV_E3_best))/
  (precision_CV_E3_best+TPR_CV_E3_best)



  out_CV_E3_best=matrix(c(TPR_CV_E3_best,FPR_CV_E3_best,
  precision_CV_E3_best,gmean_CV_E3_best,f_score_CV_E3_best),
  nrow=1,ncol=5,byrow=TRUE)
  #first row is for 1se,seconf is for BL
  #First col is TPR,second col is FPR and so on
  rownames(out_CV_E3_best) <- c("Minimising AIC (BL RULE) (E3)")
  colnames(out_CV_E3_best) <- c('Sensitivity','FPR',
  'Precision','G - mean','F-score')
  tab_best=as.table(out_CV_E3_best)

  #1se
  B=E_3_cv_AIC_matrix_1se(X,n,p) # "Estimated matrix"
  estA_E3_cv_1se=ifelse(lowerTriangle(B)!=0,1,0)# Estimated classes

  TP_CV_E3_1se = sum(sum(trueA == 1 & estA_E3_cv_1se == 1))
  FP_CV_E3_1se = sum(sum(trueA == 0 & estA_E3_cv_1se == 1))

  TPR_CV_E3_1se = TP_CV_E3_1se / NP # True Positive Rate
  (Sensitivity/recall)
  FPR_CV_E3_1se= FP_CV_E3_1se / NN # False Positive Rate

  precision_CV_E3_1se = TP_CV_E3_1se/ (TP_CV_E3_1se+FP_CV_E3_1se)
  gmean_CV_E3_1se = sqrt(TPR_CV_E3_1se*(1-FPR_CV_E3_1se))
  f_score_CV_E3_1se = (2*(precision_CV_E3_1se)*(TPR_CV_E3_1se))/
```

```
  (precision_CV_E3_1se+TPR_CV_E3_1se)


  out_CV_E3_1se=matrix(c(TPR_CV_E3_1se,FPR_CV_E3_1se,
  precision_CV_E3_1se,gmean_CV_E3_1se,f_score_CV_E3_1se),
  nrow=1,ncol=5,byrow=TRUE)
  #first row is for 1se,seconf is for BL.
  #First col is TPR,second col is FPR and so on
  rownames(out_CV_E3_1se) <- c("Minimising AIC (SE RULE) (E3)")
  colnames(out_CV_E3_1se) <- c('Sensitivity','FPR',
  'Precision','G - mean','F-score')
  tab_1se=as.table(out_CV_E3_1se)

  tab=rbind(tab_best,tab_1se)

  par(mfrow=c(2,2))
  plot(network(A),label=1:p,main="Estimated using AIC - Best lambda (E3)",
  mode = "circle")
  plot(network(theta_pairs_matrix),label=1:p,main="True network",
  mode = "circle")
  plot(network(B),label=1:p,main="Estimated using AIC - 1SE (E3)",
  mode = "circle")
  plot(network(theta_pairs_matrix),label=1:p,main="True network",
  mode = "circle")

  return(tab)
}


##### BIC
K_FOLD_E3_BIC = function(X,n,p,trueA,NP,NN,theta_pairs_matrix,theta){

  #best lambda
  A=E_3_cv_BIC_matrix_best(X,n,p,theta) # "Estimated matrix"
  estA_E3_cv_best=ifelse(lowerTriangle(A)!=0,1,0)# Estimated classes

  TP_CV_E3_best = sum(sum(trueA == 1 & estA_E3_cv_best == 1))
  FP_CV_E3_best = sum(sum(trueA == 0 & estA_E3_cv_best == 1))

  TPR_CV_E3_best = TP_CV_E3_best / NP
  # True Positive Rate (Sensitivity/recall)
  FPR_CV_E3_best= FP_CV_E3_best / NN # False Positive Rate

  precision_CV_E3_best = TP_CV_E3_best/ (TP_CV_E3_best+FP_CV_E3_best)
  gmean_CV_E3_best = sqrt(TPR_CV_E3_best*(1-FPR_CV_E3_best))
  f_score_CV_E3_best = (2*(precision_CV_E3_best)*(TPR_CV_E3_best))/
  (precision_CV_E3_best+TPR_CV_E3_best)
```

```
out_CV_E3_best=matrix(c(TPR_CV_E3_best,FPR_CV_E3_best,
precision_CV_E3_best, gmean_CV_E3_best,
f_score_CV_E3_best),nrow=1,ncol=5,byrow=TRUE)
#first row is for 1se,seconf is for BL.
#First col is TPR,second col is FPR and so on
rownames(out_CV_E3_best) <- c("Minimising BIC (BL RULE) (E3)")
colnames(out_CV_E3_best) <- c('Sensitivity','FPR','Precision','G - mean',
'F-score')

tab_best=as.table(out_CV_E3_best)

#1se
B=E_3_cv_BIC_matrix_1se(X,n,p) # "Estimated matrix"
estA_E3_cv_1se=ifelse(lowerTriangle(B)!=0,1,0)# Estimated classes

TP_CV_E3_1se = sum(sum(trueA == 1 & estA_E3_cv_1se == 1))
FP_CV_E3_1se = sum(sum(trueA == 0 & estA_E3_cv_1se == 1))

TPR_CV_E3_1se = TP_CV_E3_1se / NP # True Positive Rate (Sensitivity/recall)
FPR_CV_E3_1se= FP_CV_E3_1se / NN # False Positive Rate

precision_CV_E3_1se = TP_CV_E3_1se/ (TP_CV_E3_1se+FP_CV_E3_1se)
gmean_CV_E3_1se = sqrt(TPR_CV_E3_1se*(1-FPR_CV_E3_1se))
f_score_CV_E3_1se = (2*(precision_CV_E3_1se)*(TPR_CV_E3_1se))/
(precision_CV_E3_1se+TPR_CV_E3_1se)


out_CV_E3_1se=matrix(c(TPR_CV_E3_1se,FPR_CV_E3_1se,
precision_CV_E3_1se,
gmean_CV_E3_1se,f_score_CV_E3_1se),
nrow=1,ncol=5,byrow=TRUE)
#first row is for 1se,seconf is for BL.
#First col is TPR,second col is FPR and so on
rownames(out_CV_E3_1se) <- c("Minimising BIC (SE RULE) (E3)")
colnames(out_CV_E3_1se) <- c('Sensitivity','FPR','Precision',
'G - mean','F-score')
tab_1se=as.table(out_CV_E3_1se)

tab=rbind(tab_best,tab_1se)

par(mfrow=c(2,2))
plot(network(A),label=1:p,main="Estimated using BIC - Best lambda (E3)",
mode = "circle")
plot(network(theta_pairs_matrix),label=1:p,main="True network",
mode = "circle")
plot(network(B),label=1:p,main="Estimated using BIC - 1 SE (E3)",
mode = "circle")
plot(network(theta_pairs_matrix),label=1:p,main="True network",
mode = "circle")
```

```
  return(tab)
}
```

B.2.2 Graphical Lasso Simulations

```
#################################################################
# SIMULATIONS - WITH COMPARISON OF ALL METHODS FOR E3


### perform simulation of specific n,p,delta,q combo for E2 and
give metrics as output

simulations_E3 = function(n,p,delta,q){



  ##############################
  #### Generate theta

  a = sample(c(0, 0.5), size = (p*(p-1))/2, replace = TRUE, prob = c(q,1-q))
  b = zeros(p)
  lowerTriangle(b,byrow = TRUE ) <- a
  upperTriangle(b) = lowerTriangle(b, byrow=TRUE) #for symmetry
  diag(b)=1*delta
  theta = b
  is.positive.definite(theta, tol=1e-8)
  #convert covariance matrix to correlation matrix
  theta = cov2cor(theta)
  is.positive.definite(theta, tol=1e-8)
  sigma = inv(theta) #covariance matrix
  trueA <- ifelse(lowerTriangle(theta)!=0,1,0)# classed of theta
  theta_pairs_matrix = ifelse((theta)!=0,1,0) # pairs matrix of theta
  NP = sum(sum(trueA == 1)) # No. of Positive
  NN = sum(sum(trueA == 0)) # No. of Negative

  #### Getting the set true pairs from theta:
  true_pairs=function(theta){
    true=list()
    for (i in 1:p){
      for (j in 1:p){
        if (i>j & theta[i,j]!=0){
          true = c(true,list(c(i,j)))
        }

      }
```

```
    }
    return(true)
  }


  ######################################################
  ### Simulating X


  X=mvrnorm(n, mu=rep(0,p), Sigma = sigma)




  ######################################################

  ##########LOGLIK
  ###5 fold:


  t1=K_FOLD_E3(X,5,n,p,trueA,NP,NN,theta_pairs_matrix)

  ## 10 fold:

  t2=K_FOLD_E3(X,10,n,p,trueA,NP,NN,theta_pairs_matrix)


  ##########AIC

  t3=K_FOLD_E3_AIC(X,n,p,trueA,NP,NN,theta_pairs_matrix,theta)



  ##########BIC

  t4=K_FOLD_E3_BIC(X,n,p,trueA,NP,NN,theta_pairs_matrix,theta)



  ###### FINAL TABLE FOR E3

  FINAL=rbind(t1,t2,t3,t4)

  return(FINAL)

}
```

```
### perform simulation of specific n,p,delta,q combo a specified
number of times and give average value of each of the metrics as output
ntt_E3= function(n,p,delta,q,times){
  ar <- array(0, dim=c(8, 5, times))
  b=c()
  x=c()
  data_sens=c()
  data_fpr=c()

  for (t in c(1:times)){
    ar[,,t]=simulations_E3(n,p,delta,q)
  }

  for (i in 1:8){
    for (j in 1:5){
      for (k in 1:times){
        x=c(x,ar[i,j,k])
      }
    }
  }

  for (k in seq(from=1, to=(times*8*5), by=times)){
    b=c(b,mean(x[k:(k+times-1)]))
  }

  #SENSITIVITY BOX PLOT
  for (k in seq(from=1, to=(times*8*5), by=5*times)){
    data_sens=c(data_sens,c(x[k:(k+times-1)]))
  }
  data_sens=matrix(data_sens,nrow=times,ncol=8)
  df_sens=as.data.frame(data_sens)
  par(mfrow=c(1,1))
  boxplot(df_sens,las=2,cex.axis=0.7,main="Sensitivity across E3 methods",
  ylab="SensitIvity",
  names= c("5-fold CV (BL)","5-fold CV (SE)",
  "10-fold CV (BL)","10-fold CV (SE)","AIC (BL)",
  "AIC (SE)","BIC (BL)","BIC (SE)"))

  #FPR BOX PLOT
  for (k in seq(from=1+times, to=(times*8*5), by=5*times)){
    data_fpr=c(data_fpr,c(x[k:(k+times-1)]))
  }
  data_fpr=matrix(data_fpr,nrow=times,ncol=8)
  print(data_fpr)
  df_fpr=as.data.frame(data_fpr)
  par(mfrow=c(1,1))
  boxplot(df_fpr,las=2,cex.axis=0.7,main="FPR across E3 methods",
```

```
  ylab="FPR",names= c("5-fold CV (BL)",
  "5-fold CV (SE)","10-fold CV (BL)","10-fold CV (SE)",
  "AIC (BL)","AIC (SE)","BIC (BL)","BIC (SE)"))


  b=matrix(b,nrow=8,ncol=5,byrow = TRUE)
  rownames(b) <- c("5 - fold CV (BL RULE) (E3)"
  ,"5 - fold CV (SE RULE) (E3)","10 - fold CV (BL RULE) (E3)",
                    "10 - fold CV (SE RULE) (E3)",
                    "Minimising AIC (BL RULE) (E3)",
                    "Minimising AIC (SE RULE) (E3)",
                    "Minimising BIC (BL RULE) (E3)",
                    "Minimising BIC (SE RULE) (E3)")
  colnames(b) <- c('Sensitivity','FPR','Precision','G - mean',
  "F-score")
  out=as.table(b)
  return(list(ar,out))

}

ntt_E3(500,10,4,0.9,6)
```

B.3 AIC 1SE (Best Model) - Node wise 1,2 and graphical approaches

B.3.1 Simulations

```
##################################################
############ CHOSE AIC SE AS BEST METHOD



final = function(n,p,delta,q){

  #### Generate theta
  a = sample(c(0, 0.5), size = (p*(p-1))/2, replace = TRUE, prob = c(q,1-q))
  b = zeros(p)
  lowerTriangle(b,byrow = TRUE ) <- a
  upperTriangle(b) = lowerTriangle(b, byrow=TRUE) #for symmetry
  diag(b)=1*delta
  theta = b
  is.positive.definite(theta, tol=1e-8)
  #convert covariance matrix to correlation matrix
  theta = cov2cor(theta)
  is.positive.definite(theta, tol=1e-8)
```

```
sigma = inv(theta) #covariance matrix
trueA <- ifelse(lowerTriangle(theta)!=0,1,0)# classed of theta
theta_pairs_matrix = ifelse((theta)!=0,1,0) # pairs matrix of theta
NP = sum(sum(trueA == 1)) # No. of Positive
NN = sum(sum(trueA == 0)) # No. of Negative

#### Getting the set true pairs from theta:
true_pairs=function(theta){
  true=list()
  for (i in 1:p){
    for (j in 1:p){
      if (i>j & theta[i,j]!=0){
        true = c(true,list(c(i,j)))
      }

    }

  }
  return(true)
}

true_pairs(theta)

#########################################################################
### Simulating X
X=mvrnorm(n, mu=rep(0,p), Sigma = sigma)



############################E1


estA_E1_lasso_AIC_SE=ifelse(lowerTriangle(E_1_lasso_AIC_SE_matrix(X))
!=0,1,0) # Estimated classes
TP_E1_AIC_SE = sum(sum(trueA == 1 & estA_E1_lasso_AIC_SE== 1))
FP_E1_AIC_SE = sum(sum(trueA == 0 & estA_E1_lasso_AIC_SE == 1))

TPR_E1_AIC_SE = TP_E1_AIC_SE / NP # True Positive Rate (Sensitivity/recall)
FPR_E1_AIC_SE= FP_E1_AIC_SE / NN # False Positive Rate

precision_E1_AIC_SE = TP_E1_AIC_SE/ (TP_E1_AIC_SE+FP_E1_AIC_SE)
gmean_E1_AIC_SE = sqrt(TPR_E1_AIC_SE*(1-FPR_E1_AIC_SE))
f_score_AIC_SE = (2*(precision_E1_AIC_SE)*(TPR_E1_AIC_SE))/
(precision_E1_AIC_SE+TPR_E1_AIC_SE)


out_E1_AIC_SE=matrix(c(TPR_E1_AIC_SE,FPR_E1_AIC_SE,
precision_E1_AIC_SE,gmean_E1_AIC_SE,f_score_AIC_SE),
nrow=1,ncol=5,byrow=TRUE)
```

```
#first row is for 1se,seconf is for BL
#First col is TPR,second col is FPR and so on
rownames(out_E1_AIC_SE) <- c("Minimising AIC (SE RULE) (E1)")
colnames(out_E1_AIC_SE) <- c('Sensitivity','FPR',
'Precision','G - mean','F-score')
t1=as.table(out_E1_AIC_SE)



########################## E2
estA_E2_lasso_AIC_SE=ifelse(lowerTriangle(E_2_lasso_AIC_SE_matrix(X))
!=0,1,0) # Estimated classes
TP_E2_AIC_SE = sum(sum(trueA == 1 & estA_E2_lasso_AIC_SE== 1))
FP_E2_AIC_SE = sum(sum(trueA == 0 & estA_E2_lasso_AIC_SE == 1))

TPR_E2_AIC_SE = TP_E2_AIC_SE / NP # True Positive Rate (Sensitivity/recall)
FPR_E2_AIC_SE= FP_E2_AIC_SE / NN # False Positive Rate

precision_E2_AIC_SE = TP_E2_AIC_SE/ (TP_E2_AIC_SE+FP_E2_AIC_SE)
gmean_E2_AIC_SE = sqrt(TPR_E2_AIC_SE*(1-FPR_E2_AIC_SE))
f_score_AIC_SE = (2*(precision_E2_AIC_SE)*(TPR_E2_AIC_SE))/
(precision_E2_AIC_SE+TPR_E2_AIC_SE)


out_E2_AIC_SE=matrix(c(TPR_E2_AIC_SE,FPR_E2_AIC_SE,
precision_E2_AIC_SE,gmean_E2_AIC_SE,f_score_AIC_SE),
nrow=1,ncol=5,byrow=TRUE)
#first row is for 1se,seconf is for BL
#First col is TPR,second col is FPR and so on
rownames(out_E2_AIC_SE) <- c("Minimising AIC (SE RULE) (E2)")
colnames(out_E2_AIC_SE) <- c('Sensitivity','FPR',
'Precision','G - mean','F-score')
t2=as.table(out_E2_AIC_SE)


########################## E3
#1se
B=E_3_cv_AIC_matrix_1se(X,n,p) # "Estimated matrix"
estA_E3_cv_1se=ifelse(lowerTriangle(B)!=0,1,0)# Estimated classes

TP_CV_E3_1se = sum(sum(trueA == 1 & estA_E3_cv_1se == 1))
FP_CV_E3_1se = sum(sum(trueA == 0 & estA_E3_cv_1se == 1))

TPR_CV_E3_1se = TP_CV_E3_1se / NP # True Positive Rate (Sensitivity/recall)
FPR_CV_E3_1se= FP_CV_E3_1se / NN # False Positive Rate

precision_CV_E3_1se = TP_CV_E3_1se/ (TP_CV_E3_1se+FP_CV_E3_1se)
gmean_CV_E3_1se = sqrt(TPR_CV_E3_1se*(1-FPR_CV_E3_1se))
f_score_CV_E3_1se = (2*(precision_CV_E3_1se)*(TPR_CV_E3_1se))/
```

```
  (precision_CV_E3_1se+TPR_CV_E3_1se)


  out_CV_E3_1se=matrix(c(TPR_CV_E3_1se,FPR_CV_E3_1se,
  precision_CV_E3_1se,
  gmean_CV_E3_1se,f_score_CV_E3_1se),
  nrow=1,ncol=5,byrow=TRUE)
  #first row is for 1se,seconf is for BL
  #.First col is TPR,second col is FPR and so on
  rownames(out_CV_E3_1se) <- c("Minimising AIC (SE RULE) (E3)")
  colnames(out_CV_E3_1se) <- c('Sensitivity','FPR',
  'Precision','G - mean','F-score')
  t3=as.table(out_CV_E3_1se)



  ############################
  par(mfrow=c(2,2))
  plot(network(E_1_lasso_AIC_SE_matrix(X)),label=1:p,
  main="Estimated using AIC - 1SE (E1)",mode = "circle")
  plot(network(E_2_lasso_AIC_SE_matrix(X)),label=1:p,
  main="Estimated using AIC - 1SE (E2)",mode = "circle")
  plot(network(E_3_cv_AIC_matrix_1se(X,n,p)),label=1:p,
  main="Estimated using AIC - 1SE (E3)",mode = "circle")
  plot(network(theta_pairs_matrix),label=1:p,main="True network",
  mode = "circle")



  ############################
  final_table=rbind(t1,t2,t3)
  return(final_table)


}



ntt_final= function(n,p,delta,q,times){
  ar <- array(0, dim=c(3, 5, times))
  b=c()
  x=c()
  data_sens=c()
  data_fpr=c()
  data_prec=c()
  data_gmean=c()
  data_fscore=c()
```

```
for (t in c(1:times)){
  ar[,,t]=final(n,p,delta,q)
}

for (i in 1:3){
  for (j in 1:5){
    for (k in 1:times){
      x=c(x,ar[i,j,k])
    }
  }
}

for (k in seq(from=1, to=(times*3*5), by=times)){
  b=c(b,mean(x[k:(k+times-1)]))
}


#SENSITIVITY BOX PLOT
for (k in seq(from=1, to=(times*3*5), by=5*times)){
  data_sens=c(data_sens,c(x[k:(k+times-1)]))
}
data_sens=matrix(data_sens,nrow=times,ncol=3)
df_sens=as.data.frame(data_sens)
se_sens_E1=sd(df_sens[,1])/length(df_sens[,1])
se_sens_E2=sd(df_sens[,2])/length(df_sens[,2])
se_sens_E3=sd(df_sens[,3])/length(df_sens[,3])
par(mfrow=c(1,1))
boxplot(df_sens,main="Sensitivity across AIC SE of each approach",
ylab="Sensitivity",names= c("E1","E2","E3"))


#FPR BOX PLOT
for (k in seq(from=1+times, to=(times*3*5), by=5*times)){
  data_fpr=c(data_fpr,c(x[k:(k+times-1)]))
}
data_fpr=matrix(data_fpr,nrow=times,ncol=3)
df_fpr=as.data.frame(data_fpr)
se_fpr_E1=sd(df_fpr[,1])/length(df_fpr[,1])
se_fpr_E2=sd(df_fpr[,2])/length(df_fpr[,2])
se_fpr_E3=sd(df_fpr[,3])/length(df_fpr[,3])

par(mfrow=c(1,1))
boxplot(df_fpr,main="FPR across AIC SE of each approach",
ylab="FPR",names= c("E1","E2","E3"))

#PRECISION BOX PLOT
for (k in seq(from=1+2*times, to=(times*3*5), by=5*times)){
  data_prec=c(data_prec,c(x[k:(k+times-1)]))
}
```

```
data_prec=matrix(data_prec,nrow=times,ncol=3)
df_prec=as.data.frame(data_prec)

se_prec_E1=sd(df_prec[,1])/length(df_prec[,1])
se_prec_E2=sd(df_prec[,2])/length(df_prec[,2])
se_prec_E3=sd(df_prec[,3])/length(df_prec[,3])
par(mfrow=c(1,1))
boxplot(df_prec,main="Precision across AIC SE of each approach",
ylab="Precision",names= c("E1","E2","E3"))


#G-MEAN BOX PLOT
for (k in seq(from=1+3*times, to=(times*3*5), by=5*times)){
  data_gmean=c(data_gmean,c(x[k:(k+times-1)]))
}
data_gmean=matrix(data_gmean,nrow=times,ncol=3)
df_gmean=as.data.frame(data_gmean)
se_gmean_E1=sd(df_gmean[,1])/length(df_gmean[,1])
se_gmean_E2=sd(df_gmean[,2])/length(df_gmean[,2])
se_gmean_E3=sd(df_gmean[,3])/length(df_gmean[,3])
par(mfrow=c(1,1))
boxplot(df_gmean,main="G-Mean across AIC SE of each approach",
ylab="G-Mean",names= c("E1","E2","E3"))


#F-SCORE BOX PLOT
for (k in seq(from=1+4*times, to=(times*3*5), by=5*times)){
  data_fscore=c(data_fscore,c(x[k:(k+times-1)]))
}
data_fscore=matrix(data_fscore,nrow=times,ncol=3)
df_fscore=as.data.frame(data_fscore)
se_fscore_E1=sd(df_fscore[,1])/length(df_fscore[,1])
se_fscore_E2=sd(df_fscore[,2])/length(df_fscore[,2])
se_fscore_E3=sd(df_fscore[,3])/length(df_fscore[,3])
par(mfrow=c(1,1))
boxplot(df_fscore,main="F-score across AIC SE of each approach",
ylab="F-score",names= c("E1","E2","E3"))


means=matrix(b,nrow=3,ncol=5,byrow = TRUE)
rownames(means) <-c("Minimising AIC (SE RULE) (E1) - MEAN",
"Minimising AIC (SE RULE) (E2) - MEAN",
"Minimising AIC (SE RULE) (E3) - MEAN")
colnames(means) <- c('Sensitivity','FPR','Precision','G - mean',
'F-Score')

se=matrix(c(se_sens_E1,se_sens_E2,se_sens_E3,se_fpr_E1,se_fpr_E2,
se_fpr_E3,se_prec_E1,se_prec_E2,se_prec_E3,se_gmean_E1,se_gmean_E2,
se_gmean_E3,se_fscore_E1,se_fscore_E2,se_fscore_E3),nrow=3,ncol=5)
```

```
  rownames(se) <-c("Minimising AIC (SE RULE) (E1) - SE",
  "Minimising AIC (SE RULE) (E2) - SE","Minimising AIC (SE RULE) (E3) - SE")
  colnames(se) <- c('Sensitivity','FPR','Precision','G - mean','F-Score')

  out_means=as.table(means)
  out_se=as.table(se)
  return(list(out_means,out_se))

}

ntt_final(15,15,10,0.5,50)
```

B.4 ROC Curves

B.4.1 ROC Functions

```
#################################### ROC NODE-WISE LASSO
#############################################
library('matlab')
library('gdata')
library('matrixcalc')
library('MASS')
library('pracma')
library('glasso')
library('network')
library('CVglasso')
library('glmnet')
library('MASS')

# ########################## ROC

##### Coefficient vector following the lambda which minimises the BIC
lasso_BIC=function(X,j){
  fit=glmnet(X[,-j],X[,j])
  L <- fit$nulldev - deviance(fit)
  d <- fit$df
  n <- fit$nobs
  BIC<-log(n)*d - L
  BIC_lambda=fit$lambda[which.min(BIC)]
  model=glmnet(X[,-j],X[,j],lambda=BIC_lambda)
  coef(model)
}

AUC = function(FP, TP){
  a = head(TP, -1)
  b = tail(TP, -1)
```

```
  h = diff(-FP)
  s = sum((a + b) * h) / 2
  return(s)
}

lasso_plain=function(X,j,l){
  mod = glmnet(X[,-j],X[,j],lambda=c(l))
  coef(mod)
}


##### E1
E_1_lasso_FPR_TPR= function(X,l,trueA,NP,NN){
  E=matrix(0,length(X[1,]),length(X[1,]))
  TPR_lasso = FPR_lasso = estA_E1_lasso_roc =c()

  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
          model_i = lasso_plain(X,i,l)
          model_j = lasso_plain(X,j,l)
          if (abs(model_i[j+1,1])!=0 & abs(model_j[i,1])!=0){
            E[i,j]=1
            E[j,i]=1
          }
          else {
            E[i,j]=0
            E[j,i]=0
          }
          estA_E1_lasso_roc=ifelse(lowerTriangle(E)!=0,1,0)
          TPR_lasso = sum(sum(trueA == 1 & estA_E1_lasso_roc == 1))
          / NP # True Positive Rate (Sensitivity/recall)
          FPR_lasso = sum(sum(trueA == 0 & estA_E1_lasso_roc == 1))
          / NN # False Positive Rate
        }
      }

    }



  out=matrix(c(FPR_lasso,TPR_lasso),nrow=2,ncol=1,byrow=TRUE)
  return(out)
}

roc_glasso_E1= function(X,lambdas,trueA,NP,NN){
  FPR_LASSO = TPR_LASSO = auc = c()
  #row_names=c()
```

```
  for (l in 1:length(lambdas)){
    mod=E_1_lasso_FPR_TPR(X,lambdas[l],trueA,NP,NN)
    FPR_LASSO=c(FPR_LASSO,mod[1,1])
    TPR_LASSO=c(TPR_LASSO,mod[2,1])
  }

 # plot(
 #   x = c(0, 1),
 #   y = c(0, 1),
 #   type = "n",
 #   main = "ROC (E1)",
 #   xlab = "False Positive Rate",
 #   ylab = "True Positive Rate"
 # )
 # lines(FPR_LASSO,TPR_LASSO, col = "black", lwd = 6)

  auc=AUC(FPR_LASSO,TPR_LASSO)
  return(c(auc,FPR_LASSO,TPR_LASSO))
}

#roc_glasso_E1(X,c(0.01,0.02,0.03),700,trueA,NP,NN)
#lambda = seq(0.01,0.2,by=0.001)


##### E2

E_2_lasso_FPR_TPR= function(X,l,trueA,NP,NN){
  E=matrix(0,length(X[1,]),length(X[1,]))
  TPR_lasso = FPR_lasso = estA_E2_lasso_roc =c()

  for (j in 1:length(X[1,])){
    for (i in 1:length(X[1,])){
      if (i>j){
        model_i = lasso_plain(X,i,l)
        model_j = lasso_plain(X,j,l)
        if (abs(model_i[j+1,1])!=0 | abs(model_j[i,1])!=0){
          E[i,j]=1
          E[j,i]=1
        }
        else {
          E[i,j]=0
          E[j,i]=0
        }
        estA_E2_lasso_roc=ifelse(lowerTriangle(E)!=0,1,0)
        TPR_lasso = sum(sum(trueA == 1 & estA_E2_lasso_roc == 1))
        / NP # True Positive Rate (Sensitivity/recall)
        FPR_lasso = sum(sum(trueA == 0 & estA_E2_lasso_roc == 1))
        / NN # False Positive Rate
      }
```

```
    }

  }



  out=matrix(c(FPR_lasso,TPR_lasso),nrow=2,ncol=1,byrow=TRUE)
  return(out)
}

roc_glasso_E2= function(X,lambdas,trueA,NP,NN){
  FPR_LASSO = TPR_LASSO = auc = c()
  #row_names=c()

  for (l in 1:length(lambdas)){
    mod=E_2_lasso_FPR_TPR(X,lambdas[l],trueA,NP,NN)
    FPR_LASSO=c(FPR_LASSO,mod[1,1])
    TPR_LASSO=c(TPR_LASSO,mod[2,1])
  }
 #
 # plot(
 #   x = c(0, 1),
 #   y = c(0, 1),
 #   type = "n",
 #   main = "ROC (E2)",
 #  xlab = "False Positive Rate",
 #   ylab = "True Positive Rate"
 # )
 # lines(FPR_LASSO,TPR_LASSO, col = "black", lwd = 6)

  auc=AUC(FPR_LASSO,TPR_LASSO)
  return(c(auc,FPR_LASSO,TPR_LASSO))
}

lambda = seq(0.001,0.2,by=0.001)




#################################### ROC GRAPHICAL LASSO
#####################################################

E_3_glasso_FPR_TRP = function(X,l,trueA,NP,NN,p){

  estA_glasso = rep(0,length=(p*(p-1)/2))
  TPR_GLASSO = FPR_GLASSO = c()
  gla <- glasso(cov(X),l)
  P <- gla$wi
```

```
  estA_glasso <- ifelse(abs(lowerTriangle(P))!=0,1,0)
  TPR_GLASSO = sum(sum(trueA == 1 & estA_glasso == 1)) / NP
  # True Positive Rate
  FPR_GLASSO = sum(sum(trueA == 0 & estA_glasso == 1)) / NN
  # False Positive Rate

  out=matrix(c(FPR_GLASSO,TPR_GLASSO),nrow=2,ncol=1,byrow=TRUE)
  return(out)
}


roc_glasso_E3= function(X,lambdas,trueA,NP,NN,p){
  auc=rep(0,length(lambdas))
  FPR_GLASSO = TPR_GLASSO =c()
  #row_names=c()

  for (l in 1:length(lambdas)){
    mod=E_3_glasso_FPR_TRP(X,lambdas[l],trueA,NP,NN,p)
    FPR_GLASSO=c(FPR_GLASSO,mod[1,1])
    TPR_GLASSO=c(TPR_GLASSO,mod[2,1])
  }

  #     plot(
  #      x = c(0, 1),
  #      y = c(0, 1),
  #      type = "n",
  #      main = "ROC (E3)",
  #      xlab = "False Positive Rate",
  #      ylab = "True Positive Rate"
  #    )
  # lines(FPR_GLASSO,TPR_GLASSO, col = "black", lwd = 6)

  auc=AUC(FPR_GLASSO,TPR_GLASSO)
  return(c(auc,FPR_GLASSO,TPR_GLASSO))
}
```

B.4.2 ROC Simulations

```
################################################ simulations


roc_simulations =function(n,p,delta,q,lambdas){

  a = sample(c(0, 0.5), size = (p*(p-1))/2, replace = TRUE, prob = c(q,1-q))
  b = zeros(p)
  lowerTriangle(b,byrow = TRUE ) <- a
  upperTriangle(b) = lowerTriangle(b, byrow=TRUE) #for symmetry
  diag(b)=1*delta
  theta = b
  is.positive.definite(theta, tol=1e-8)
  #convert covariance matrix to correlation matrix
  theta = cov2cor(theta)
  #print(is.positive.definite(theta, tol=1e-8))
  sigma = inv(theta) #covariance matrix
  trueA = ifelse(lowerTriangle(theta)!=0,1,0)# classes of theta
  theta_pairs_matrix = ifelse((theta)!=0,1,0) # pairs matrix of theta
  NP = sum(sum(trueA == 1)) # No. of Positive
  NN = sum(sum(trueA == 0)) # No. of Negative


  #### Getting the set true pairs from theta:
  true_pairs=function(theta){
    true=list()
    for (i in 1:p){
      for (j in 1:p){
        if (i>j & theta[i,j]!=0){
          true = c(true,list(c(i,j)))
        }

      }

    }
    return(true)
  }


  ##################################################

  ### Simulating X


  X=mvrnorm(n, mu=rep(0,p), Sigma = sigma)
```

```
  FPR1=roc_glasso_E1(X,lambdas,trueA,NP,NN)[2:(length(lambdas)+1)]
  FPR2=roc_glasso_E2(X,lambdas,trueA,NP,NN)[2:(length(lambdas)+1)]
  FPR3=roc_glasso_E3(X,lambdas,trueA,NP,NN,p)[2:(length(lambdas)+1)]

  TPR1=roc_glasso_E1(X,lambdas,trueA,NP,NN)
  [(length(lambdas)+2):(2*length(lambdas)+1)]
  TPR2=roc_glasso_E2(X,lambdas,trueA,NP,NN)
  [(length(lambdas)+2):(2*length(lambdas)+1)]
  TPR3=roc_glasso_E3(X,lambdas,trueA,NP,NN,p)
  [(length(lambdas)+2):(2*length(lambdas)+1)]


#AUCC
  AUC1=roc_glasso_E1(X,lambdas,trueA,NP,NN)[1]
  AUC2=roc_glasso_E2(X,lambdas,trueA,NP,NN)[1]
  AUC3=roc_glasso_E3(X,lambdas,trueA,NP,NN,p)[1]

  auc_out=c(AUC1,AUC2,AUC3)
  out=matrix(c(FPR1,FPR2,FPR3,TPR1,TPR2,TPR3),nrow=6,
  ncol=length(lambdas),byrow=TRUE)

  return(list(out,auc_out))
}



ntt_roc = function(n,p,delta,q,lambdas,times){

  FPR1=FPR2=FPR3=TPR1=TPR2=TPR3=matrix(0,nrow = times,
  ncol = length(lambdas))
  a <- array(0, dim=c(6, length(lambdas), times))
  auc_matrix= matrix(0,nrow=times,ncol=3)

  for (t in c(1:times)){
    l=unlist(roc_simulations(n,p,delta,q,lambdas)[1])
    l=matrix(l,nrow=6,ncol=length(lambdas))
    a[,,t]=l
    FPR1[t,]=a[1,,t]
    FPR2[t,]=a[2,,t]
    FPR3[t,]=a[3,,t]
    TPR1[t,]=a[4,,t]
    TPR2[t,]=a[5,,t]
    TPR3[t,]=a[6,,t]

    #auc
    auc_matrix[t,]=unlist(roc_simulations(n,p,delta,q,lambdas)[2])

  }
```

```
Av_FPR1=colMeans(FPR1)
Av_FPR2=colMeans(FPR2)
Av_FPR3=colMeans(FPR3)
Av_TPR1=colMeans(TPR1)
Av_TPR2=colMeans(TPR2)
Av_TPR3=colMeans(TPR3)

auc_df=as.data.frame(auc_matrix)
boxplot(auc_df,names=c('AUC E1','AUC E2','AUC E3'),
main="Area under the curve across approaches")

plot(
  x = c(0, 1),
  y = c(0, 1),
  type = "n",
  main = "ROC (E1)",
  xlab = "False Positive Rate",
  ylab = "True Positive Rate"
)
lines(Av_FPR1,Av_TPR1, col = "black", lwd = 6)

av_auc_e1=AUC(Av_FPR1,Av_TPR1)

plot(
  x = c(0, 1),
  y = c(0, 1),
  type = "n",
  main = "ROC (E2)",
  xlab = "False Positive Rate",
  ylab = "True Positive Rate"
)
lines(Av_FPR2,Av_TPR2, col = "black", lwd = 6)

av_auc_e2=AUC(Av_FPR2,Av_TPR2)



plot(
  x = c(0, 1),
  y = c(0, 1),
  type = "n",
  main = "ROC (E3)",
  xlab = "False Positive Rate",
  ylab = "True Positive Rate"
)
lines(Av_FPR3,Av_TPR3, col = "black", lwd = 6)

av_auc_e3=AUC(Av_FPR3,Av_TPR3)
```

```
  return(list(av_auc_e1,av_auc_e2,av_auc_e3))

}

ntt_roc(5,10,4,0.1,seq(0.01,0.06,by=0.001),2)
```